Politecnico di Torino

Politecnico di Torino

Master degree in
Electronic Engineering
A.a. 2023/2024
Sessione di Laurea Luglio 2024

Microcontroller-based implementation of a data extraction algorithm from a radar signal

Relatori:
Eros Gian Alessandro Pasero
Marina Mondin
Fereydoun Daneshgaran

Candidati:
Gaia Pia Pistillo

Summary

CHAPTER 1

Introduction

Precise and low-cost localization is fundamental in a variety of applications. The focus of the current research is on augmented reality for multitude of applications from gaming to training of personnel in various industries. In such applications, one key requirement is that of identifying a precise position on a given object, possibly in a complex scenario, like a cluttered environment, with a lot of background noise (one example could be that of identifying a precise location on a jet engine when training a maintenance operator with augmented reality). A typical approach when performing localization is that of using trilateration, starting from range information that has been collected through three or more radiofrequency receivers.

The process can be facilitated by using an active target, that generates a unique return signal used to unambiguously identify the target. A further choice that must be made is the selection of the considered radiofrequency transceiver. To minimize the implementation cost, a low-complexity chirp RADAR (i.e., a Frequency Modulated Continuous Wave RADAR), can be used to derive the range information.

Having to identify the signal reflected by a target in a cluttered environment, the system uses an active target equipped with a small and low-cost transponder that generates, in response to the RADAR signal, a bandpass signal that can be separated from the clutter which is generally located around baseband in the down-converted RADAR signal. It is therefore necessary to have a specific demodulator that recovers this bandpass signal generated by the active target and processes the signal to generate the range information needed for trilateration.

Given this general scenario, the thesis work has been focused on the study, simulation, and implementation of the demodulator able to recover the bandpass signal generated in the down-converted RADAR signal by the active target in response to the RADAR chirp signal.

More specifically, this thesis aims to implement a data extraction algorithm from a RADAR signal on a microcontroller for target localization.
The data extraction algorithm is based on an envelope detector; by demodulating the

received signal from a chirp RADAR, the target range can be calculated from the estimated beat frequency. Then, the target can be precisely localized by using the trilateration method which consists of combining the target ranges calculated from three receivers' signals.

The thesis outlines the theory behind this implementation, presents the simulation results and details the system implementation.

In particular, in Chapter 2 the trilateration method is described and in Chapter 3 the characteristics of a FMCW RADAR with chirp modulation are presented. In Chapter 4 the block diagram for the data extraction algorithm is introduced and the individual blocks are described, with a particular focus on FIR filters which are at the basis of this implementation. In Chapter 5 the hardware implementation on a TI C2000 F28379D microcontroller is described. In Chapter 6 the algorithm performances are analyzed. Conclusions are drawn in Chapter 7.

CHAPTER 2

Trilateration

The final application related to this thesis work is that of locating a target. The complexity in locating an object is represented by the fact that it is not easy to directly find the position of an object in space; most sensors, usually, estimate the distances from the object. For instance, sonars and RADARs emit electromagnetic waves and calculate the distance by detecting how long it takes for the signal to come back after being reflected by the object.

One of the most common techniques to locate an object is trilateration, which calculates the position of an object given several distance measurements.

This chapter provides an overview of the trilateration technique and its implementation.

2.1 Principle of trilateration

To find the position of an object in a two-dimensional space using trilateration, the position of at least three reference points must be known. The position of the object can be found as the intersection of three circumferences each having as center one of the reference points and as radius the measured distance of one of the reference points from the object.

This minimum requirement can be explained by using the geometrical interpretation of the technique.

2.2 Geometrical interpretation

Indicating the unknown target position as P and the known positions of the three reference points as L_1, L_2 and L_3, trilateration can be explained by a geometrical point of view as follows.

By measuring the distance from the target of one reference point, L_1, a circumference with radius equal to the measured distance, d_1, and center equal to L_1 can be individuated; the target P is on this circumference (fig. 2.2.1).

By measuring the distance, d_2, from the target position to the second reference point, L_2, a second circumference, with radius d_2 and center L_2, can be individuated; the first and

second circumferences intersect in two points, as can be seen in Figure 2.2.2 , thus a third circumference is needed to obtain just one point.

The third circumference can be drawn with radius d_3, i.e distance between L_3 and P, and center L_3: the point of intersection between the three circumferences is the location P, as shown in Figure 2.2.3.

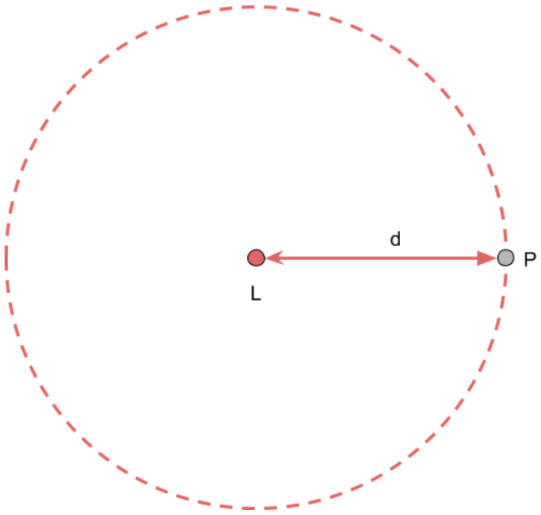

Figure 2.2.1: Circumference with radius d and center L

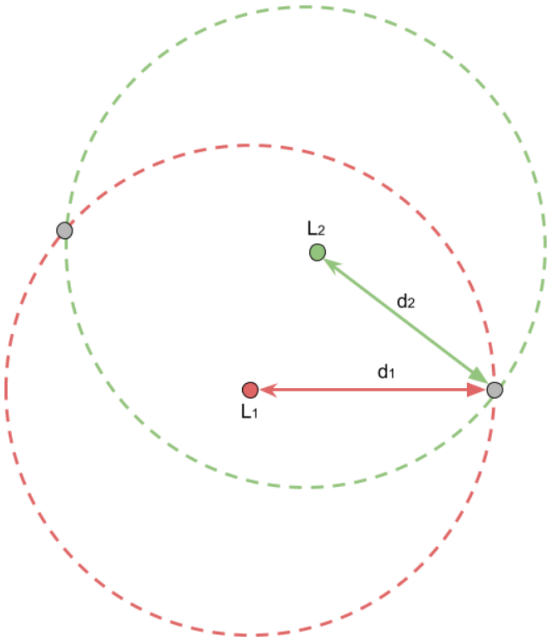

Figure 2.2.2: Intersections between circumference with radius d_1 and center L_1 and radius d_2 and center L_2

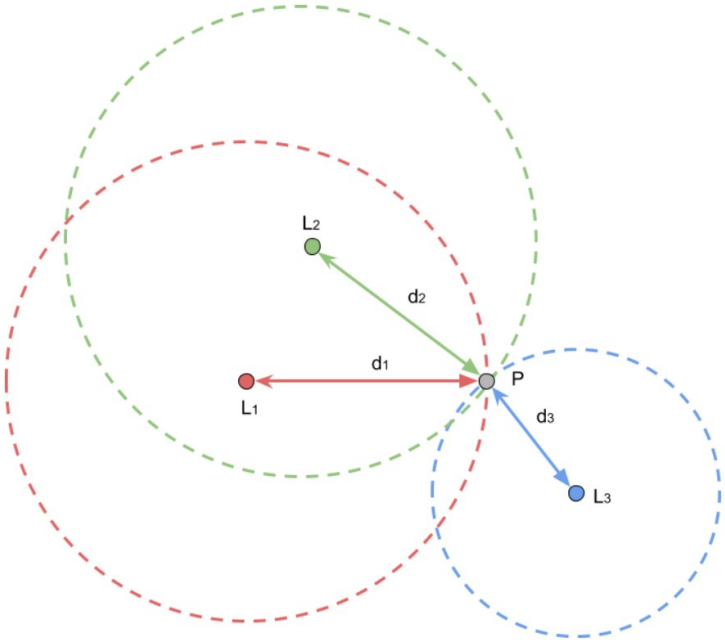

Figure 2.2.3: Trilateration problem in 2D

2.3 Mathematical interpretation

The trilateration problem in two-dimensional space from a mathematical point of view consists in solving the set of three equations corresponding to the three circumferences described above.

A point (x,y) in the Cartesian plane lies on a circumference with radius r centered at (c_x, c_y) if and only if is a solution to this equation:

$$(x - c_x)^2 + (y - c_y)^2 = {d_1}^2$$

Thus, having the three reference points L_1, L_2, L_3 each, respectively, with coordinates $(x_1, y_1), (x_2, y_2), (x_3, y_3)$ and distance from the target of d_1, d_2, d_3, the expressions of the the circumferences having as radius d_i and center L_i are:

$$(x - x_1)^2 + (y - y_1)^2 = {d_1}^2$$

$$(x - x_2)^2 + (y - y_2)^2 = {d_2}^2$$

$$(x - x_3)^2 + (y - y_3)^2 = {d_3}^2$$

Then the location of the target with coordinates (x,y) can be obtained by solving the 3 equations above simultaneously.

This approach works in an ideal case in which the distance measurements are taken with extremely high accuracy. In practice, however, the signals are usually influenced by perturbations, as in the case of indoor localization. As a result, the intersection of the three circumferences will be a domain instead of a point [1], as shown in Figure 2.3.1. The intersection domain D is defined by the following set of equations:

$$(x - x_1)^2 + (y - y_1)^2 \leq {d_1}^2$$

$$(x - x_2)^2 + (y - y_2)^2 \leq {d_2}^2$$

$$(x - x_3)^2 + (y - y_3)^2 \leq {d_3}^2$$

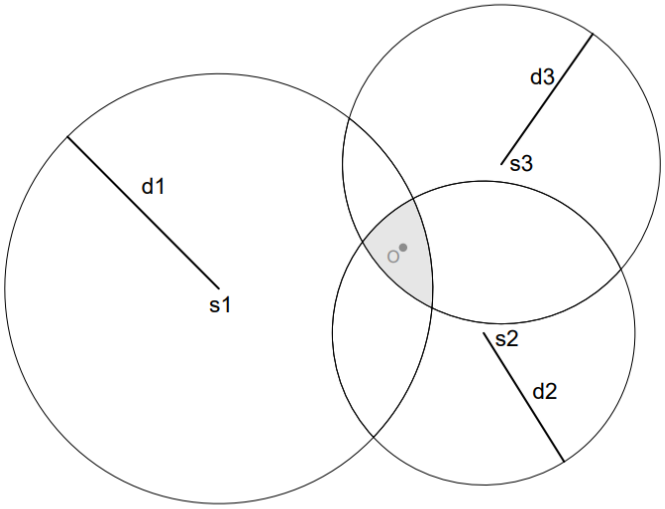

Figure 2.3.1: Indoor Localization Problems in a 2D Space

2.4 Optimization algorithm

The mathematical approach is not the best solution for solving the trilateration problem. As said before, high accurate measurements are needed; in the worst case, if the measurements are wrong the three circumferences do not intersect and the set of equations does not have a perfect solution. Moreover, the complexity of that approach increases if more than three reference points are taken into consideration; this is the case of the localization problem in three-dimensional space, and an example is the one of GPS, which requires four reference points.

Thus, a better approach is the one in which an optimization algorithm is used: the goal is finding the point which is the best approximation to the actual position of the target object.

One of the most commonly employed algorithms is the least squares estimation method, which minimizes the sum of the squares of the differences between the measured distances and the calculated distances from the reference points to the unknown point. Examples are presented in [2], [3].

Another possibility consists of minimizing the mean square error, i.e. the average squared deviation between the measured and predicted values.

Indicating with P the position of the target object and with L_i the position of the reference points which have a distance d_i from the target, with i=[1,2,3], the goal is to find a point X which has the same distance d_i from the reference points. The point X is the point which minimizes a certain error function, in this case the mean squares error between measured distances, d_i, and the calculated distances, *dist(X, L_i),* from the reference points to the unknown point. In detail, the errors to minimize are three, one for each reference point. The errors can be expressed as follows:

$$e_1 = d_1 - dist(X, L_1)$$

$$e_2 = d_2 - dist(X, L_2)$$

$$e_3 = d_3 - dist(X, L_3)$$

And the three errors can be merged into one contribution by averaging their squares, obtaining the mean squared error (MSE):

$$MSE = \frac{\sum_{i=1}^{N}[d_i - dist(X, L_i)]^2}{N}$$

CHAPTER 3

RADAR

In this thesis work the range measurements are obtained using a Frequency Modulated Continuous Wave (FMCW) RADAR with chirp modulation.

This chapter provides first an overview of RADARs and then a description of FMCW RADARs.

3.1 RADAR

A RADAR, i.e. RAdio Detection And Ranging, is a system used for the detection and the range measurement of a target, Figure 3.1.1. RADAR operates by transmitting radio waves toward an object and then detecting the waves reflected from the object. It is composed of a transmitter, which produces electromagnetic waves in the radio/microwave domain, and a receiver, which receives the echo signal reflected by the target.

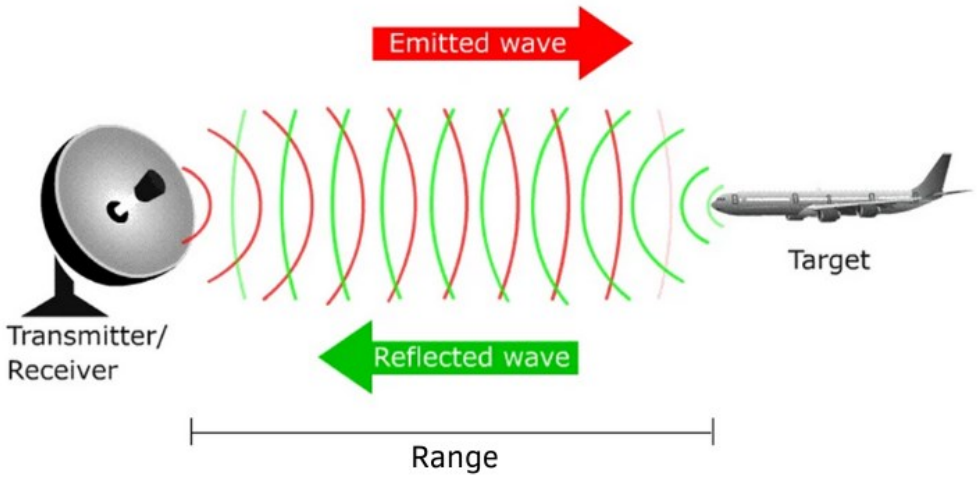

Figure 3.1.1: RADAR localization

There are two primary types of RADAR: one emitting pulsing waves and one emitting continuous waves. Pulse RADAR systems transmit a series of equally spaced short and high-energy pulses and calculate the distance to the target by measuring the time delay between a transmitted pulse and the returning reflected signal; to do this the transmission of the pulses is alternated with a period of no transmission in which the eco can be received. Continuous wave RADAR, or CW RADAR, radiates a transmitting power without

interruption. The echo signal is received and processed simultaneously and continuously. CW RADARs are divided in:

1. unmodulated CW RADAR: the transmitted signal is constant in amplitude and frequency; by observing the Doppler shift, the shift in frequency between the transmitted and received signal due to the target's velocity relative to the RADAR; moving objects in the detection range of the sensor are detected but it is not possible to measure distances with them;
2. FMCW RADAR: the transmitted signal is constant in amplitude but modulated in frequency; having a frequency that varies in time makes it possible to measure a difference between the received signal frequency and the transmitted signal frequency. This frequency difference is called beat frequency and from it the range can be derived.

3.1.1 Range measurements

In the case of pulse RADAR, the range is measured by considering the time of flight of one pulse:

$$R = \frac{c_0 \cdot \Delta t}{2}$$

where R is the range, $c_0 = 3 * 10^8\, m/s$ is the velocity of the wave in air, and Δt is the time it takes for the transmitted signal to be reflected by the object and come back to the receiver.

In the case of FMCW RADAR, the distance is derived from the frequency. First, the received and transmitted signals are mixed; then the obtained signal is filtered to isolate the low-frequency component given by the difference between the transmitted and received frequency, i.e. the beat frequency; in the end, the range is derived from the beat frequency.

Assuming that the transmitted signal is:

$$x_1(t) = cos(2\pi f_1 t + \alpha_1)$$

and the received signal is:

$$x_2(t) = cos(2\pi f_2 t + \alpha_2)$$

where α_i is the initial phase of the signal,

the mixing of the two signals can be done by multiplying them and obtaining the superimposition of two sinusoids, one with a higher frequency given by the sum of f_1 and f_2 and one with a lower frequency given by the difference between f_1 and f_2.

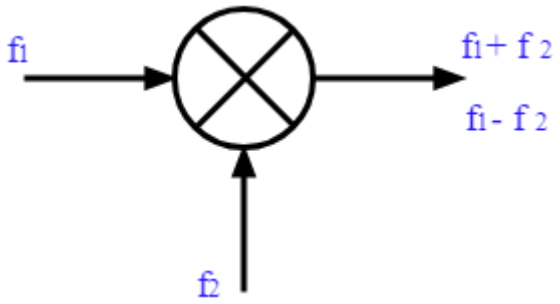

Figure 3.1.2: Frequency mixer

The result of the mixing is:

$$x_1(t) * x_2(t) = cos(2\pi f_1 t + \alpha_1) * cos(2\pi f_2 t + \alpha_2) =$$

$$= \frac{1}{2} * [cos(2\pi t(f_1 + f_2) + \alpha_1 + \alpha_2) + cos(2\pi t(f_1 - f_2) + \alpha_1 - \alpha_2)]$$

Then, by using a low-pass filter the low-frequency component can be extracted and by performing a frequency demodulation of the signal the information for deriving the range is obtained.

3.2 FMCW RADAR

There are some advantages of FMCW RADAR to pulse RADAR. For instance, the sensitivity provided by FMCW technology is more than 30 times higher than that of pulsed RADAR transmitters, which maximizes signal strength and enables it to deliver superior measurement reliability with a greater signal-to-noise ratio SNR (the ratio of the power of a signal to the power of background noise).

The basic features of FMCW RADAR are:

1. Ability to measure very small ranges to the target (the minimal measured range is comparable to the transmitted wavelength);
2. Ability to measure simultaneously the target range and its relative velocity;
3. Very high accuracy of range measurement;

4. Signal processing after mixing is performed at a low-frequency range, considerably simplifying the realization of the processing circuits;
5. Safety from the absence of pulse radiation with a high peak power.

3.2.1 Frequency modulation and Modulation patterns

Frequency Modulation is a modulation in which the carrier wave's frequency is altered according to the instantaneous amplitude of the modulating signal, keeping phase and amplitude constant, Figure 3.2.1.

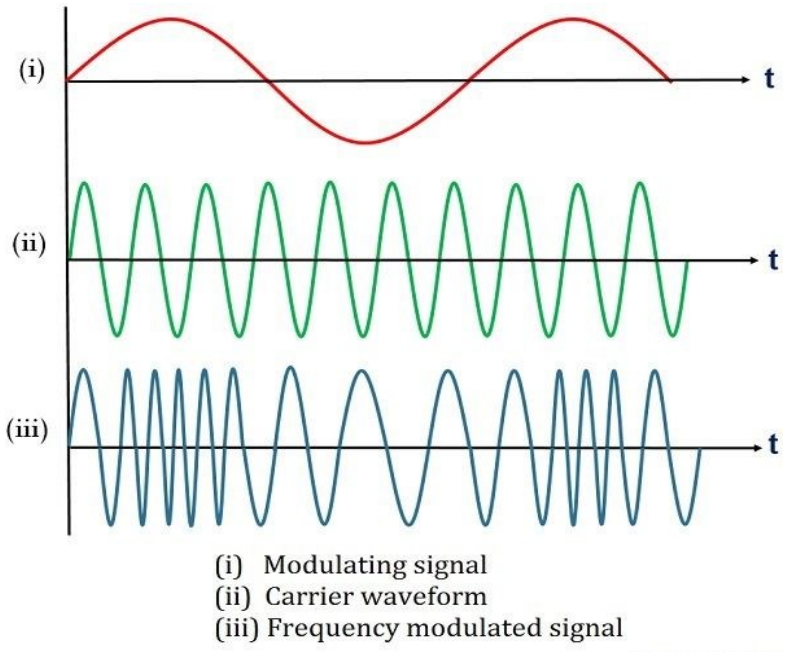

Figure 3.2.1: Frequency modulation

Starting with a carrier waveform with constant frequency f_0, amplitude A, and initial phase α expressed as:

$$x(t) = A * cos(2\pi f_0 t + \alpha)$$

and a modulating signal $s(t)$, the frequency modulated signal can be expressed as:

$$x(t) = A * cos(2\pi \cdot g(s(t)) \cdot t + \alpha)$$

There are several patterns for modulating the transmitted signal in frequency, Figure 3.2.2:

1. Sawtooth modulation: this modulation pattern is used in a relatively large range (maximum distance) combined with a negligible influence of Doppler frequency (for example, a maritime navigation RADAR);
2. Triangular modulation: this modulation allows easy separation of the difference frequency Δf of the Doppler frequency f_d;
3. Square-wave modulation (simple frequency-shift keying, FSK): this modulation is used for a very precise distance measurement at close range by phase comparison of the two echo signal frequencies. It has the disadvantage, that the echo signals from several targets cannot be separated from each other, and that this process enables only a small unambiguous measuring range;
4. Stepped modulation (staircase voltage): this is used for interferometric measurements and expands the unambiguous measuring range.

The most commonly used signal for frequency modulation in a FMCW RADAR is the chirp signal that will be described in the next paragraph.

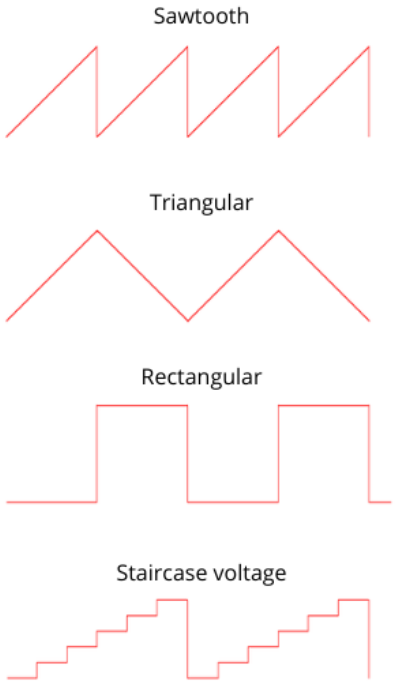

Figure 3.2.2: Waveforms for frequency modulation

3.2.2 FMCW RADAR with chirp modulation

With chirp modulation, the signal frequency increases or decreases linearly over time.

A CHIRP (Compressed HIgh Resolution Pulse) signal is a signal in which the frequency varies linearly with time; with an up-chirp the frequency increases linearly while with a down-chirp the frequency decreases linearly, Figure 3.2.3.

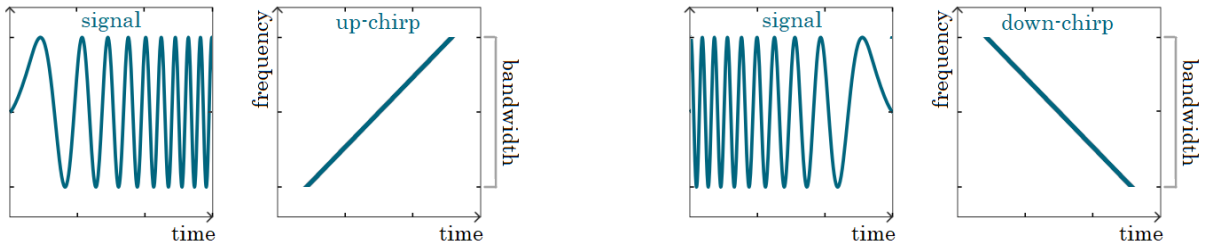

Figure 3.2.3: Up-chirp and down-chirp signal in time and frequency domain

The instantaneous frequency f(t) for an up-chirp signal can be expressed as:

$$f(t) = f_0 + kt$$

where f_0 is the starting frequency and k is the chirp rate, defined as $k = \frac{BW}{T}$, where BW is the bandwidth and T is the duration of the chirp.

Taking into account this, the transmitted RADAR signal frequency modulated with a up-chirp signal can be represented as:

$$x_1 = A_1 * cos(2\pi(f_0 t + \frac{k}{2}t^2))$$

where A_1 is the amplitude of the signal.

Due to the time needed for the signal to arrive at the target and come back to the RADAR receiver, the received signal is delayed by a quantity $\Delta t = \frac{2R}{c}$ where R is the range and c is the speed of light in air and can be expressed as follows:

$$x_2 = A_2 * cos(2\pi(f_0(t - \Delta t) + \frac{k}{2}(t - \Delta t)^2))$$

At the receiver, the received signal will be mixed with the transmitted signal generating a beat frequency f_b, which is the frequency difference between the transmitted and received signal. Due to the relation between the beat frequency f_b, the time delay Δt, and the chirp rate k given by the following equation:

$$f_b = k * \Delta t$$

the range R can be computed as:

$$R = \frac{c f_b}{2k}$$

CHAPTER 4

Schematic block diagram

In this chapter the system localization scheme is presented and the schematic block diagram of the data extraction algorithm is described.

4.1 System localization scheme

The system localization scheme operates using a frequency modulated continuous wave FMCW RADAR signal, based on the use of a chirp signal. A chirp signal is basically a frequency modulated signal that uses as information signal a periodic triangular wave, that, within each period linearly increases the frequency of the transmitted signal. The FMCW signal is transmitted and intercepted by an active transponder at distance d. The transponder returns an amplitude modulated and amplified version of the RADAR signal that it intercepts. The RADAR receiver operates by mixing the received signal with the original transmitted one. The result has two signal components: one at twice the carrier frequency and one at a lower frequency, that contains the signal of interest. The signal of interest is centered around the transponder modulation frequency, denoted as f_0, and contains information about the distance d from the active transponder. In particular, the distance information is contained in the "beat" frequency f_m due to frequency difference in the FMCW chirp signal due to the double delay introduced by the transponder at distance d. We can assume that $f_0 \gg f_m$. The components around DC also contain range information about all the obstacles on the path of the RADAR that generate a return signal. We will denote this DC components as "clutter". The signal of interest, i.e. the received, frequency demodulated and low-passed signal (that we will consider as our input signal), within each period of the periodic triangular wave, can be written as

$$y(t) = (A + (A_m cos(2\pi f_m t + \theta)) * cos(2\pi f_0 t + \theta_0) + c(t)$$

where $c(t)$ is the clutter signal.

Our goal will be that of recovering the frequency f_m, that will allow us to determine the distance d and then perform trilateration (using multiple transponders that operate according to the same principle). The algorithm used to obtain an estimate $\widehat{f_m}$ of the frequency f_m will be analyzed, simulated and implemented on a microcontroller.

4.2 Schematic block diagram

The schematic block diagram of the algorithm for the envelope detection of the RADAR signal is shown in figure 4.2.1.

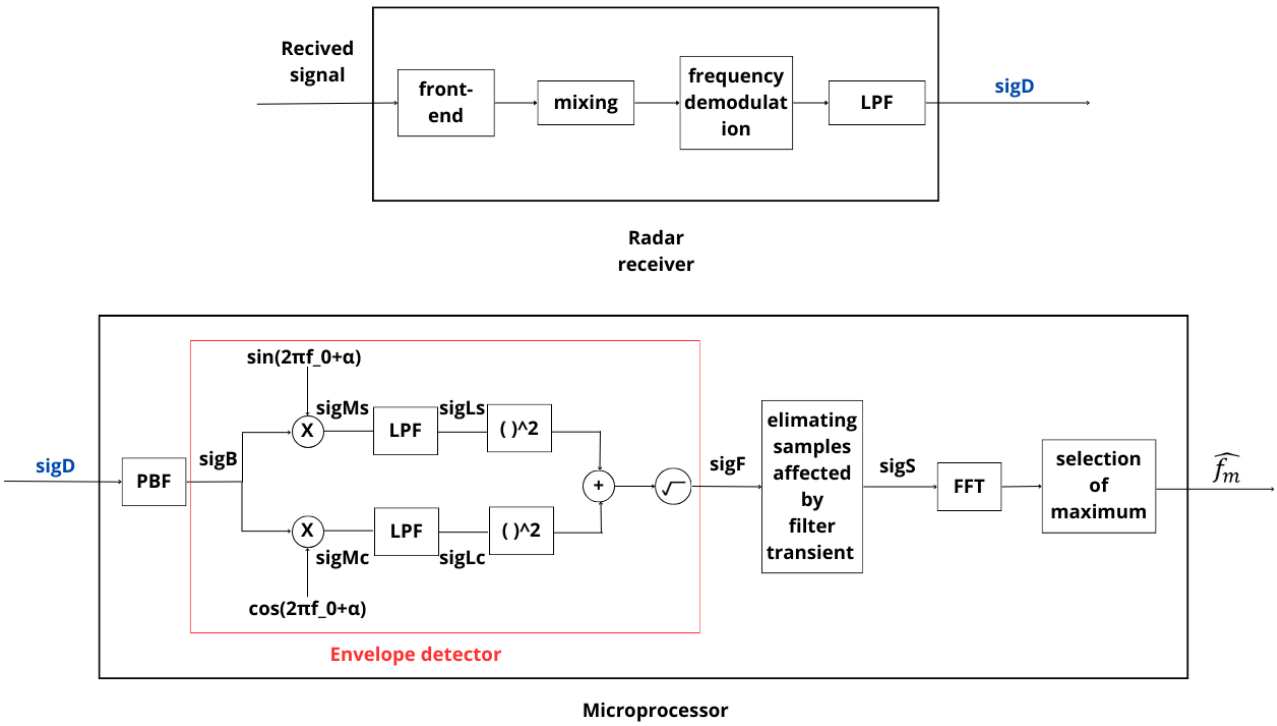

Figure 4.2.1: Schematic block diagram

A dataset of real received RADAR data has been collected and used for testing, but in order to test the performance of the developed algorithm a simulator for the generation of the signal has also been created in MATLAB.

The function for the generation of the simulated signal generates a random signal with baseband scattering with bandwidth B, amplitude modulation with carrier with amplitude A at frequency f_0 and sidelobes at $f_0 + f_m$ and $f_0 - f_m$ with amplitude A_m.

The signal received back by the RADAR has approximately this form:

$$y(t) = (A + (A_m cos(2\pi f_m t + \theta)) * cos(2\pi f_0 t + \theta_0) + c(t)$$

where:

1. c(t) is the clutter, that is the signal reflected by the environment;
2. $Acos(2\pi f_m t + \theta)$ is the signal at frequency f_m, the beat frequency, which contains the range information;
3. $cos(2\pi f_0 t + \theta_0)$ is the modulating signal;
4. A is a constant.

To obtain this in MATLAB the signal is generated in the following way:

$$signal = (A + Am * cos(2 * pi * fm * n * Ts + pha))'.* cos(2 * pi * f0 * n * Ts + pha0)'$$

$$+Samp * scatter(201:end, 1) + Namp * noise2(201:end, 1);$$

To take into account the imperfections of the real signal, the phases *pha* and *pha0* are generated randomly while the frequencies f_m and f_0 are generated by adding to the known values of f_m, 750 Hz, and f_0, 200 kHz, random values. Moreover, scattering in the baseband and noise to the whole signal are added.

An example of the signal generated during a MATLAB simulation is shown in Figure 4.2.2:

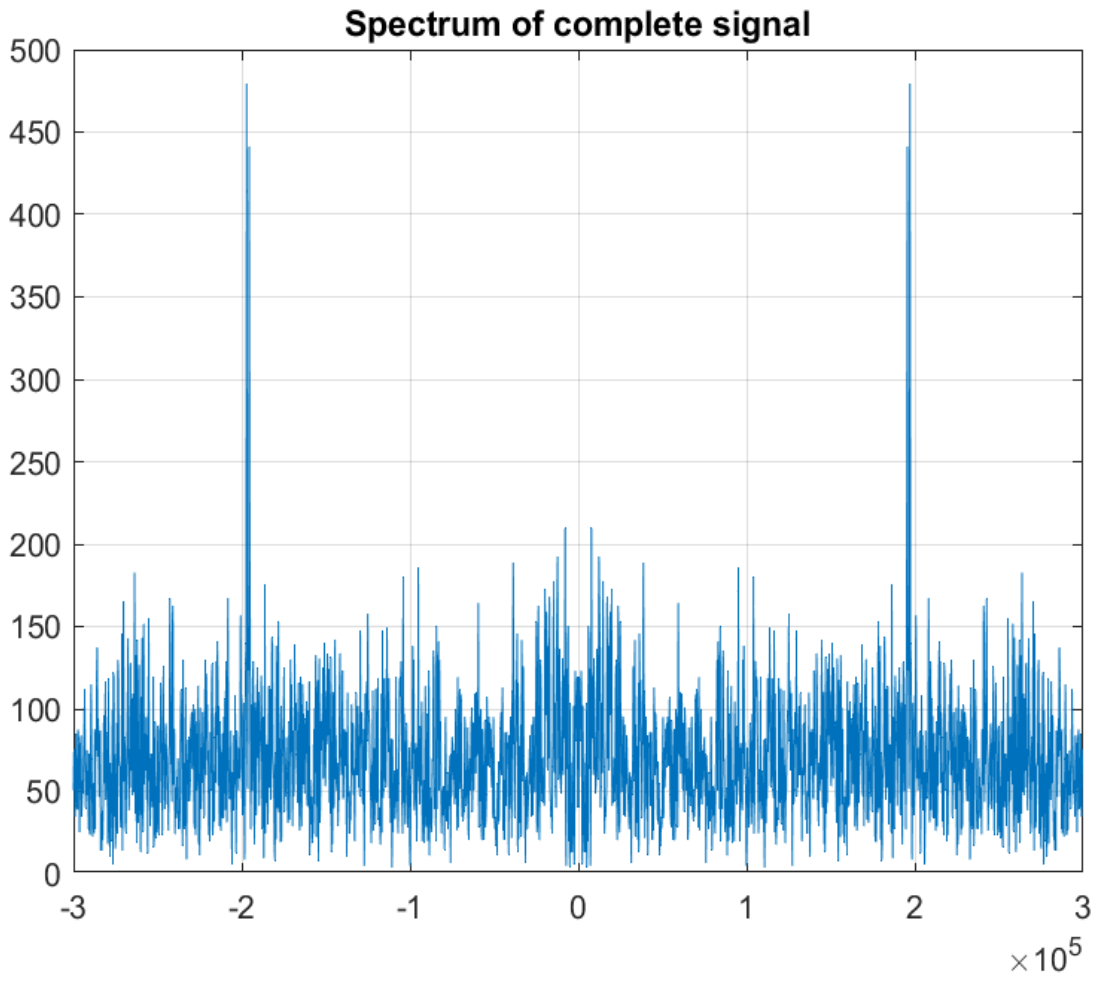

Figure 4.2.2: Complete signal

In the baseband there is noise representing the clutter. At frequencies f_0 and -f_0 there is the modulated signal with sidelobes at $f_0 + f_m$ and $f_0 - f_m$. The modulated signal can be seen better by zooming around frequency f_0 (figure 4.2.3).

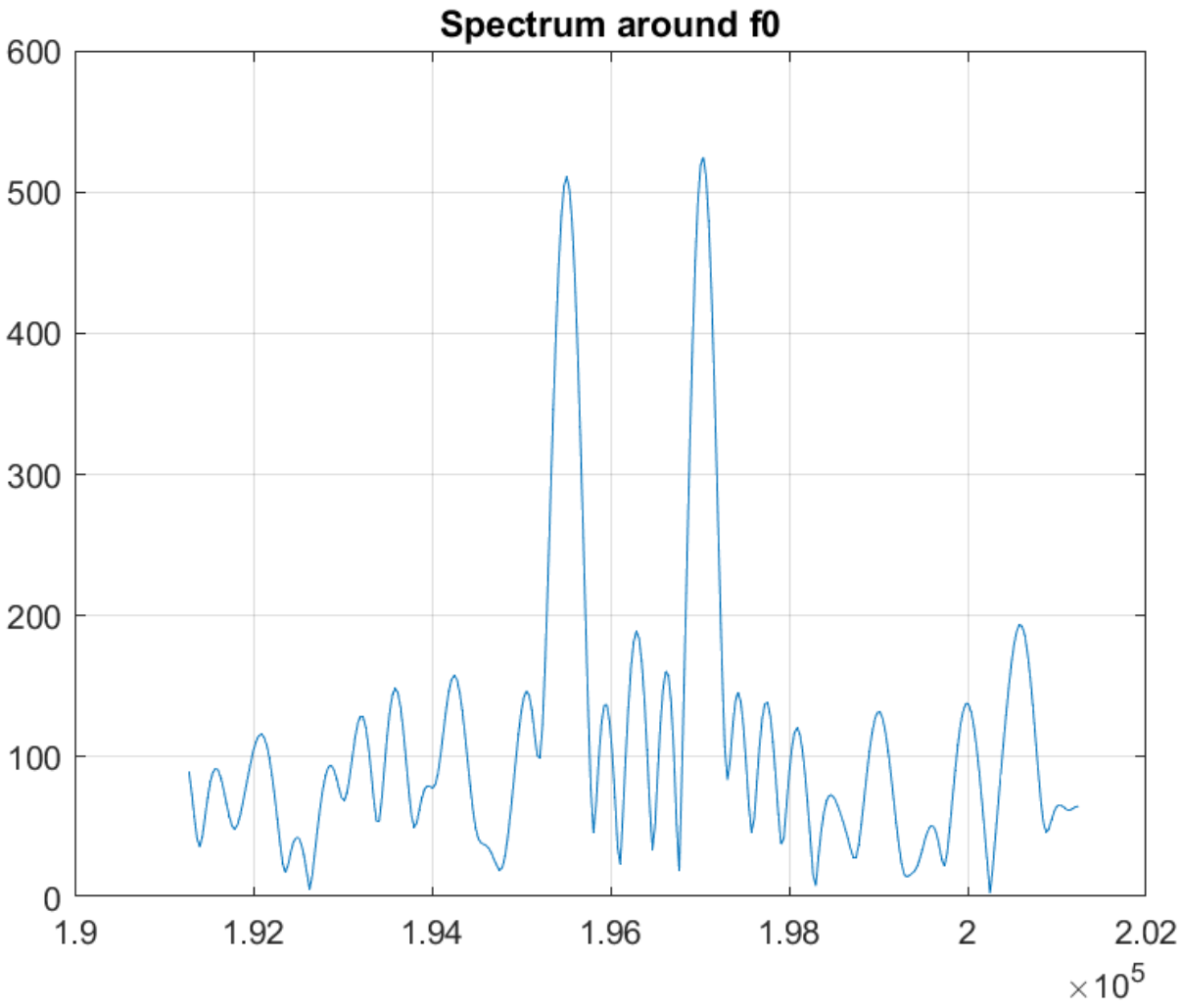

Figure 4.2.3: Complete signal around f_0

As can be observed the modulated signal is a sinusoid centered at frequency f_0.

Considering that the RADAR sends both the chirp signal and the timing signal only a block of the signal, corresponding to the samples between two rising edges of the timing signal, is analyzed. Thus, the signal processed by the algorithm is sigD, which can be seen in Figure 4.2.4 and zoomed around f_0 in Figure 4.2.5.

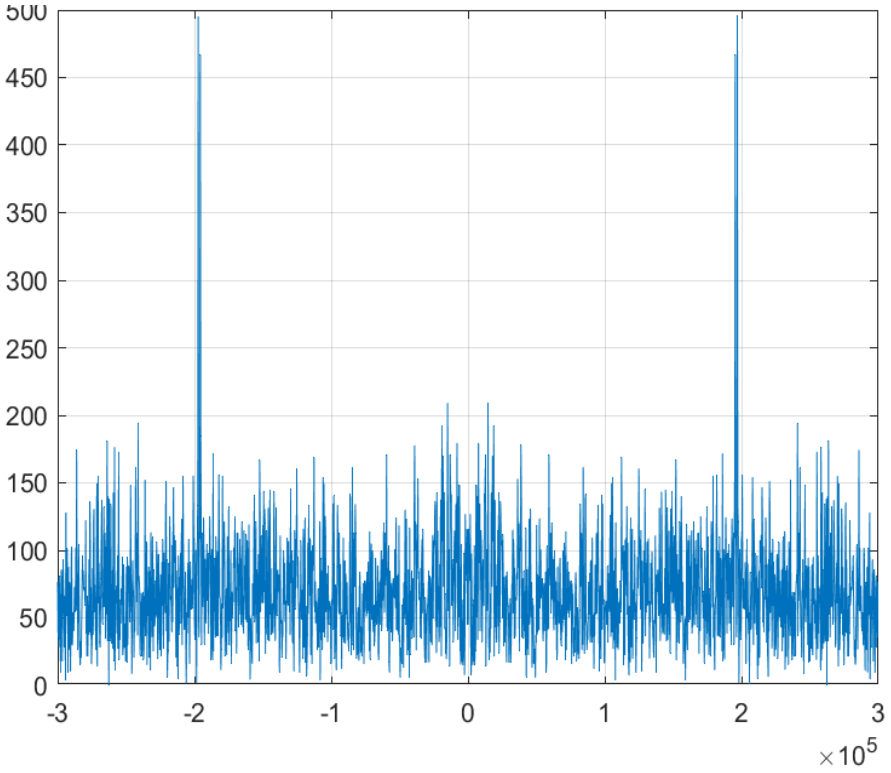

Figure 4.2.4: Spectrum of one block

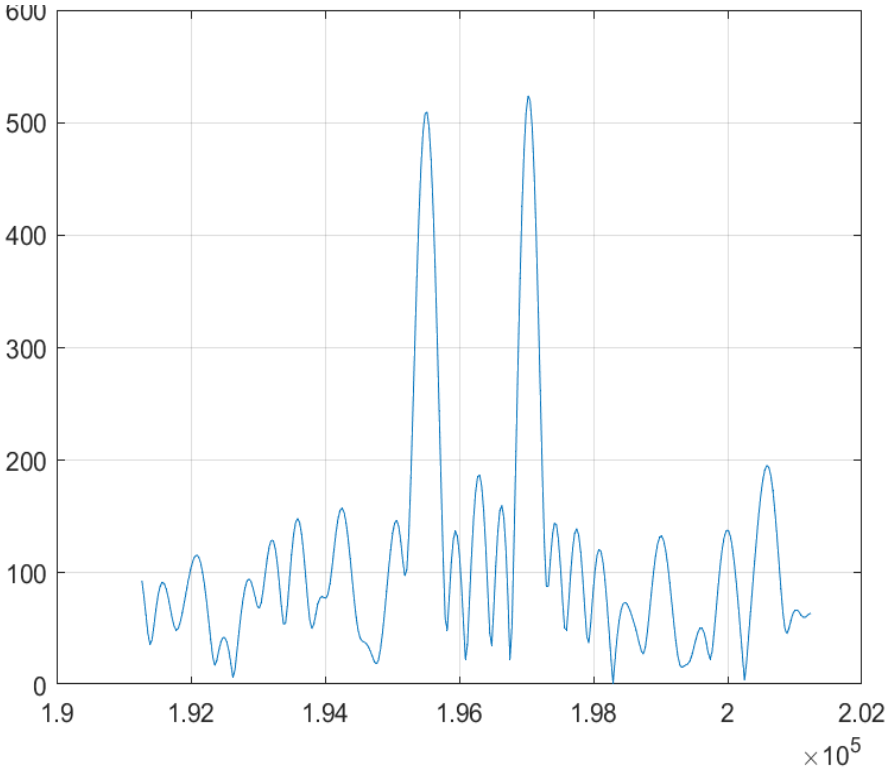

Figure 4.2.5: sigD around f_0

To eliminate the clutter in the baseband sigD is filtered around frequency f_0 with a passband filter obtaining sigB, Figure 4.2.6; a zoom of sigB around f_0 can be seen in Figure 4.2.7.

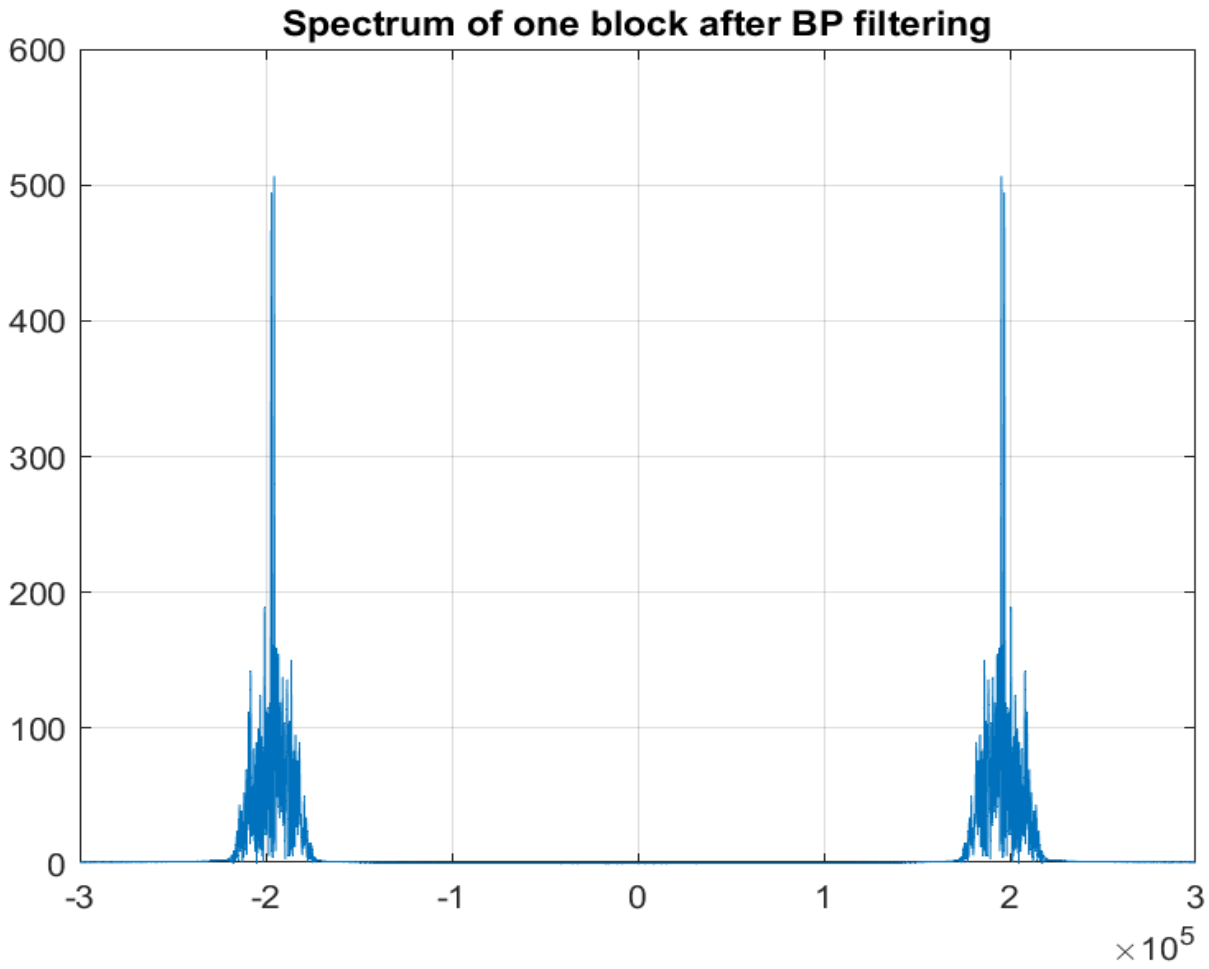

Figure 4.2.6: sigB

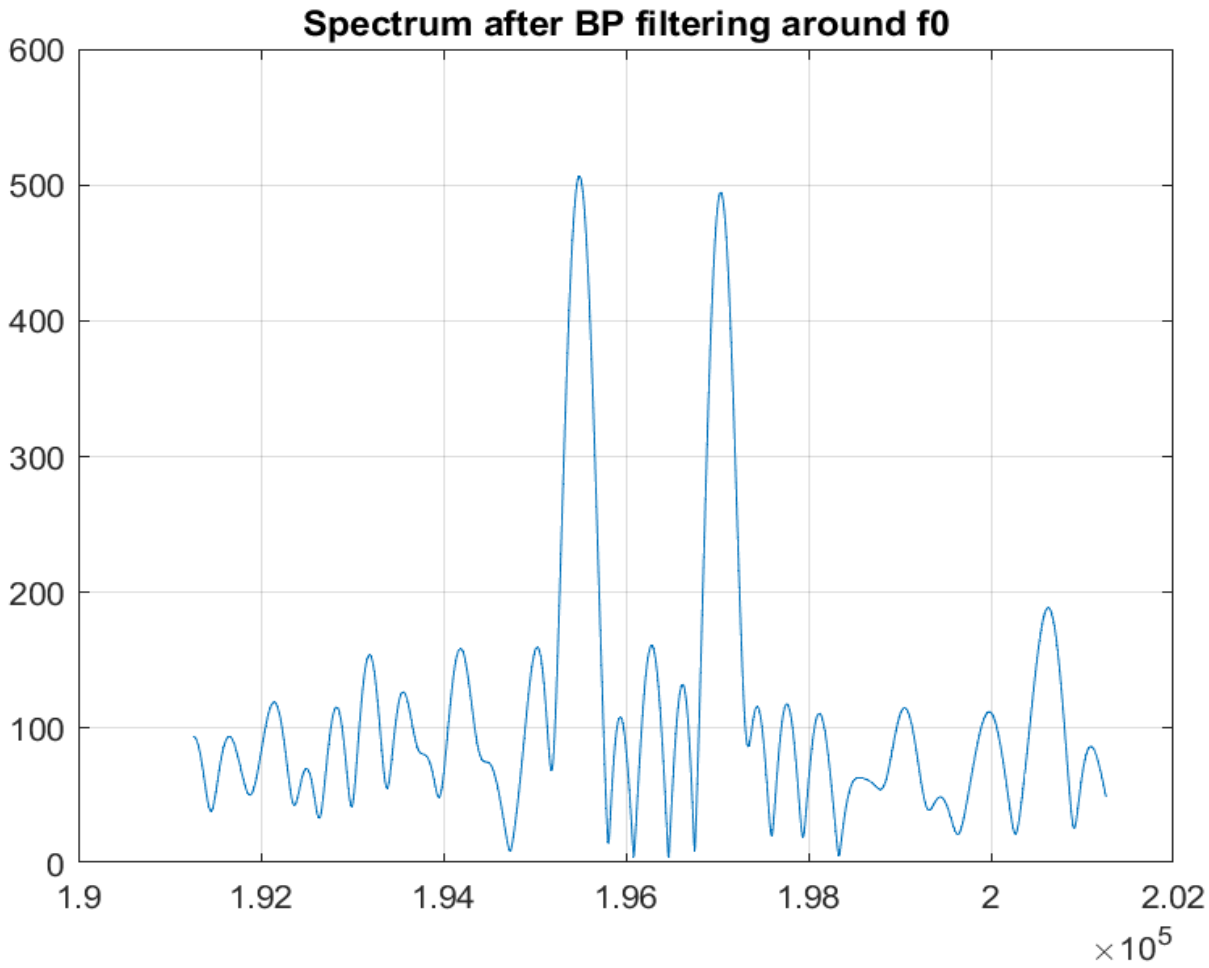

Figure 4.2.7: sigB around f0

SigB has the form:

$sigB(t) = x(t) * cos(2\pi f_0 t + \theta_0)$.

where x(t) is:

$x(t)=A + (A_m cos(2\pi f_m t + \theta)$

Then, to demodulate the signal an envelope detector is used.

First, sigMc and sigMs are obtained by multiplying sigB for a cosine and sine function, respectively:

$$sigMc(t) = x(t) * cos(2\pi f_0 t + \theta_0) * cos(2\pi f_0 t + \alpha)$$

$$sigMc(t) = x(t) * cos(2\pi f_0 t + \theta_0) * sin(2\pi f_0 t + \alpha)$$

Secondly, sigMc and sigMs are filtered with a lowpass filter obtaining sigLc and sigLs:

$$sigLc = x(t) * cos(\theta_0 - \alpha)\frac{1}{2}$$

$$sigLc = x(t) * sin(\theta_0 - \alpha)\frac{1}{2}$$

Then, to isolate x(t) sigLc and sigLs are squared and then summed together:

$$(sigLc)^2 + (sigLs)^2 = (\frac{1}{2}x(t))^2 cos^2(\theta_0 - \alpha) + (\frac{1}{2}x(t))^2 sin^2(\theta_0 - \alpha) = (\frac{1}{2}x(t))^2$$

As the last step of the demodulation, sigF is obtained by doing the square root of the result of the previous sum:

$$sigF(t) = \sqrt{(\frac{1}{2}x(t))^2} = \frac{1}{2}|x(t)| = \frac{1}{2}|A + (A_m cos(2\pi f_m t + \theta)|$$

Then, sigS is obtained by eliminating the first del+1 samples of sigF, where del is the delay introduced by the filters, which corresponds to $\frac{L_1}{2} + \frac{L_2}{2}$ with Li equal to the order of the filter.

As can be seen in figure 1.7 sigF is a sinusoid with frequency $2f_m$.

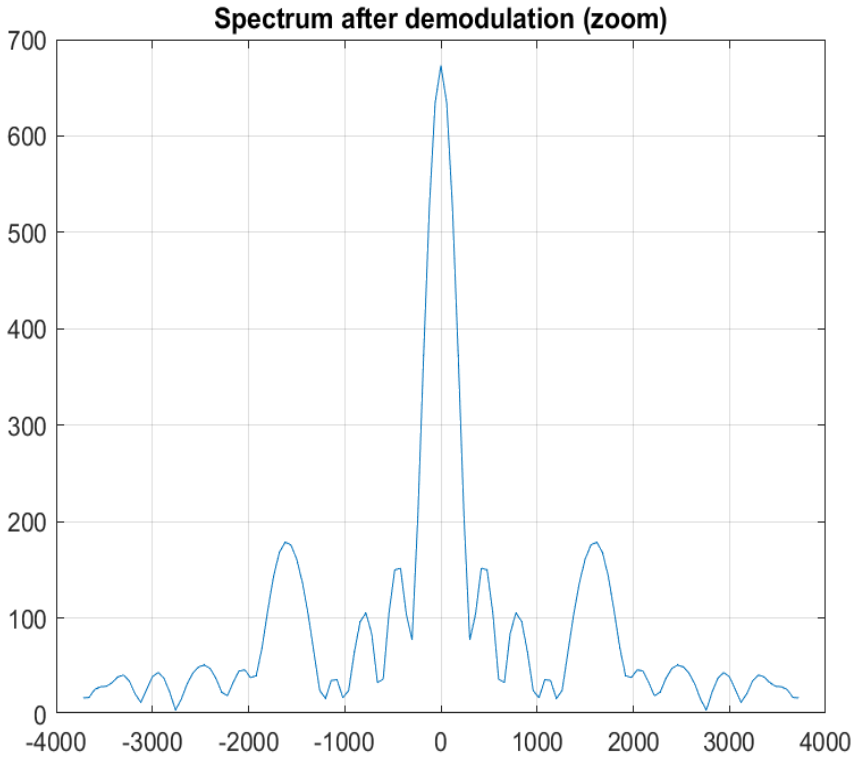

Figure 1.7: sigF

The frequency f_m can be estimated following these steps:

1. eliminating the DC component of sigS, which corresponds to the mean value of sigS:

$$sigS1 = sigS - mean(sigS);$$

2. looking for the maximum of the FFT of sigS1.

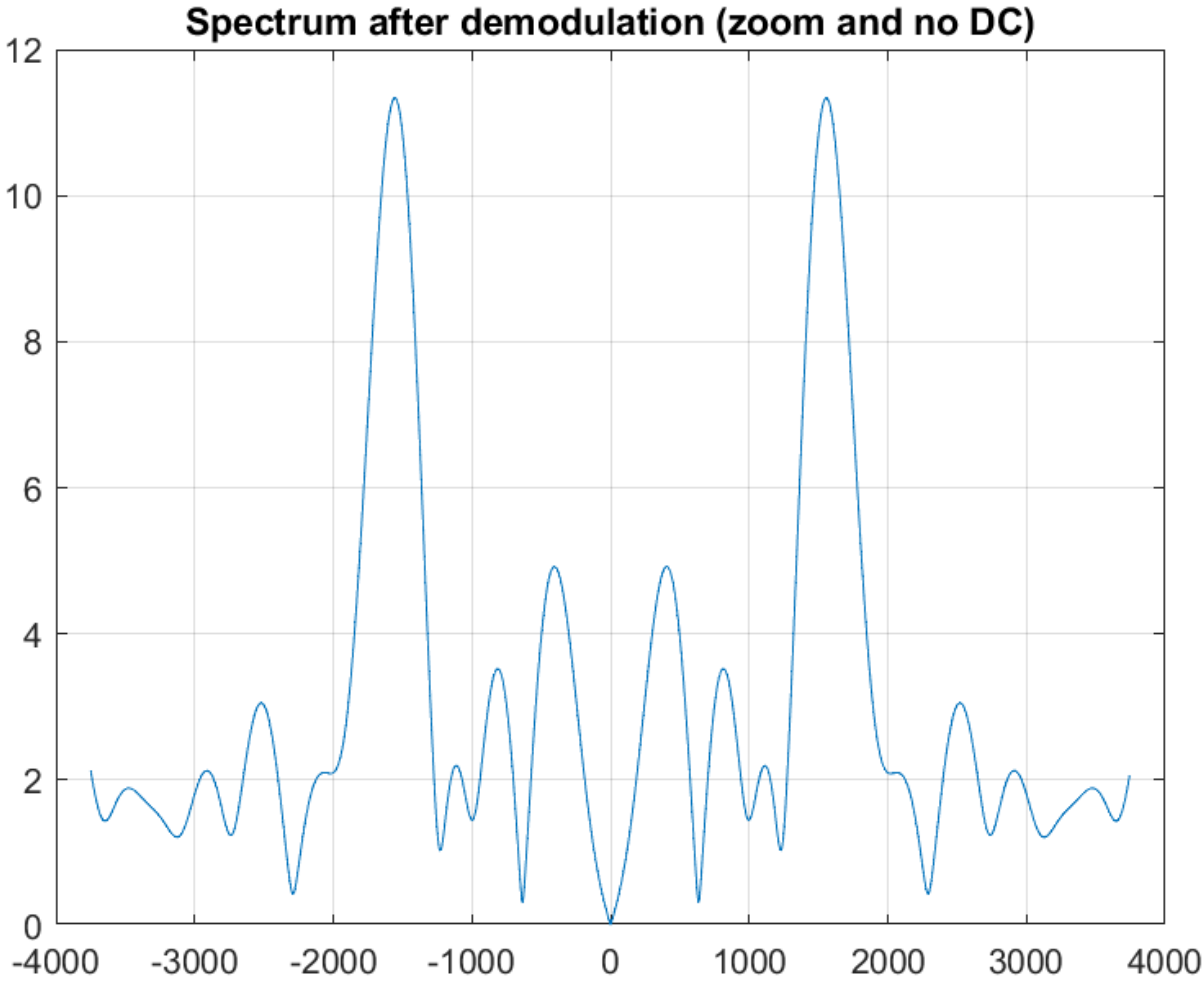

Figure 1.8: sigS1

CHAPTER 5

HARDWARE IMPLEMENTATION

In this chapter, the implementation of the envelope detector algorithm on the TI TMS320F2837xD Microcontroller is described.

5.1 Microcontroller

For the hardware implementation of the envelope detector, a TI TMS320F2837xD Dual-Core Microcontroller has been chosen.

The TMS320F2837xD is a powerful 32-bit floating-point microcontroller unit (MCU) designed for advanced closed-loop control applications such as industrial motor drives; solar inverters and digital power; electrical vehicles and transportation; and sensing and signal processing.

The dual real-time control subsystems are based on TI's 32-bit C28x floating-point CPUs, which provide 200 MHz of signal processing performance in each core.

The most relevant features of the microcontroller are listed below:

1. CLA (Control Law Accellerator)

The F2837xD microcontroller family features two CLA real-time control coprocessors. The CLA is an independent 32-bit floating-point processor that runs at the same speed as the main CPU. The CLA responds to peripheral triggers and executes code concurrently with the main C28x CPU. This parallel processing capability can effectively double the computational performance of a real-time control system. By using the CLA to service time-critical functions, the main C28x CPU is free to perform other tasks, such as communications and diagnostics

2. MEMORY

The TMS320F2837xD supports up to 1MB (512KW) of onboard flash memory with error correction code (ECC) and up to 204KB (102KW) of SRAM. Two 128-bit secure zones are also available on each CPU for code protection.

3. PERIPHERALS

Performance analog and control peripherals are also integrated on the F2837xD MCU to further enable system consolidation. Four independent 16-bit ADCs provide precise and

efficient management of multiple analog signals, which ultimately boosts system throughput. The new sigma-delta filter module (SDFM) works in conjunction with the sigma-delta modulator to enable isolated current shunt measurements. The Comparator Subsystem (CMPSS) with windowed comparators allows for protection of power stages when current limit conditions are exceeded or not met. Other analog and control peripherals include DACs, PWMs, eCAPs, eQEPs, and other peripherals.

To implement and test our application the F28379D LaunchPad has been chosen, Figure 5.1.1, which has USB connected isolated XDS100v2 JTAG debug probe for real-time debug and flash programming, 4x 20-pin headers/connectors, and programmable buttons and LEDs.

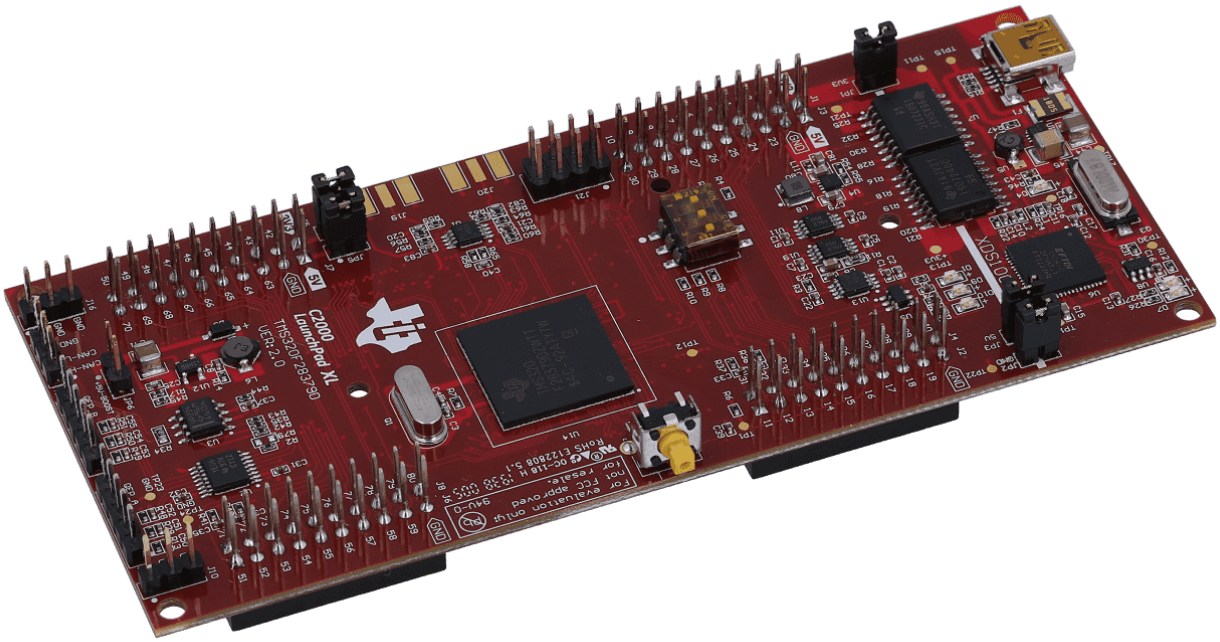

Figure 5.1.1: F28379D LaunchPad

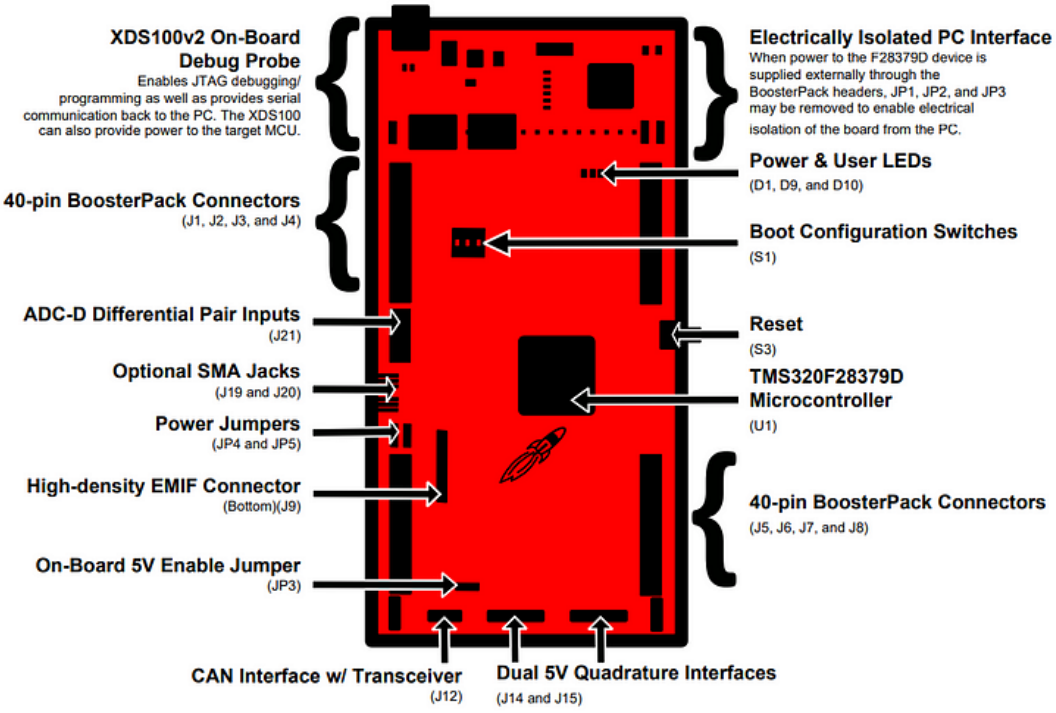

Figure 5.1.2: LAUNCHXL-F28379D Board Overview

5.2 Implementation

The goal was to load the developed code for the envelope detector implementation on the flash memory of the microcontroller to obtain a stand-alone operation. The needed data, which are the acquired RADAR signal data and the filter coefficients, have also been loaded on the flash memory.

5.2.1 Data acquisition

The data processed by the detector have been acquired and frequency demodulated by the RADAR board, and are further processed by the detector in order to estimate the distance between the RADAR transmitter and the active target.

The coefficients for the pass-band filter and the low pass filter have been generated using the *firpm* function in Matlab which uses the Parks-McClellan algorithm.

All the data have been loaded on the flash memory of the CPU1 of the microcontroller by programming it using the F021 Flash API library, composed by functions written, compiled and validated by Texas Instruments.

5.2.3 Flash programming

On the F28376D devices, each CPU has its own flash bank [256KB (128KW)], the total flash for each device is 512KB (256KW). Each bank is divided in 7 sectors. Only one bank can be programmed or erased at a time and the code to program the flash should be executed out of RAM. Each bank is divided in 7 sectors.

Moreover, the Flash module contains a Flash state machine (FSM) to perform program and erase operations.

A typical flow to program the Flash consists of erasing, programming and verifying the selected sector.

To erase a given sector the *Fapi_issueAsyncCommandWithAddress()* command is used which takes as parameters the command to issue the FSM, which for erasing is *Fapi_Erasesector*, and the Flash sector address.

Then to program and verify the *Fapi_issueProgrammingCommand()* command is used which takes as parameters:

1. the start address in Flash for the data to be programmed;
2. the pointer to the address of the Data buffer containing the data that needed to be written in the sector;
3. the number of 16-bit words in the Data buffer;
4. the pointer to the ECC buffer address, 0 if the ECC buffer is not being programmed;
5. the number of 8-bit bytes in the ECC buffer, 0 if the ECC buffer is not being programmed;
6. the programming mode to use, *Fapi_DataOnly* to only program the data buffer.

Instead, to read data from the flash into a user-given buffer the *Fapi_doMarginRead()* command is used which takes as parameters:

1. the start address for region to read;
2. the address of buffer to return read data;
3. the length of region in 32-bit words to read;
4. the read mode, Fapi_NormalRead.

5.3 Code implementation

The code written for the microcontroller is in C language. In the following paragraphs the basic theory and the code behind the implementation of the main elements of the implemented algorithm will be explained.

5.3.1 Filter implementation

In the implementation of the envelope detector, there are two filters, a pass band filter and a low pass filter, both are designed as even symmetric FIR filters with an odd number of coefficients.

FIR FILTER

In signal processing, a filter is a system that alters an incoming signal in the desired way to extract useful information and discard undesirable components.

There are two main classes of filters, which differ in the type of signals processed: analog and digital. Digital filters, being implemented in software algorithms, are more advantageous compared to analog ones, which are implemented with circuital components [6]: they do not suffer from component tolerances, their response is invariant to temperature and time, and they are insensitive to electrical noise to a great extent. Moreover, they are programmed easily on digital hardware and are very versatile in the desired responses they can produce. Considering these advantages, digital filters are preferred to reduce noise or obtain certain aspects of the given signal; this is why they are used in applications like data compression, biomedical signal processing, speech and image processing, data transmission, digital audio, and telephone echo cancellation.

The common representation of digital filters through their impulse response leads to their classification in FIR, Finite Impulse Response, and IIR, Infinite impulse response; as the name underlines FIR filters have a finite duration pulse response which means having a nonzero pulse response for only a finite number of samples, while IIR filters have an infinite duration pulse response and therefore an infinite number of nonzero samples in an impulse response.

For the localization system developed in this thesis, even symmetry FIR filters with an odd number of coefficients have been used, which have linear phase and a constant delay equal to an integer number of samples.

The output of a FIR filter, y[n], is given by the discrete-time convolution between the input signal and the impulse response of the filter, which can be written as:

$$y[n] = \sum_{k=0}^{M-1} h[k] * x[n-k];$$

where x[n] is the input signal, h[k] is the impulse response of the filter, and M is the number of filter coefficients/taps. For each output calculated, there are M multiplications, as shown in Figure 5.3.1.

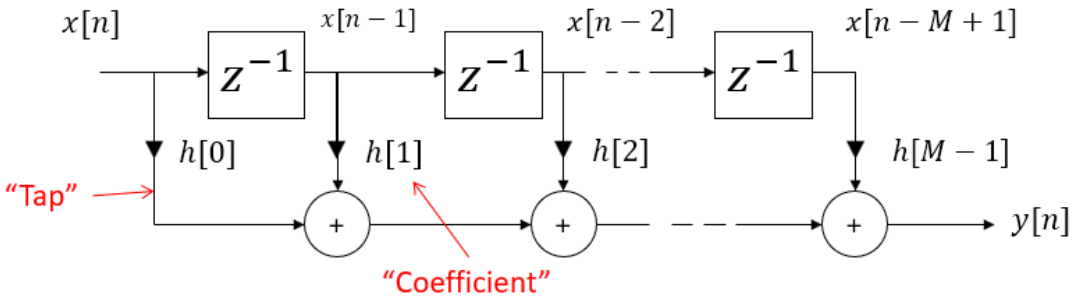

Figure 5.3.1: FIR filter

If the filter is symmetric, i.e. if its impulse response satisfies the following condition:

$h[n] = h[M-1-n]$, with n=1,2,3,..,M-1

the number of multiplications for each output sample reduces to (M-1)/2+1 (when M is odd), as shown in Figure 5.3.2.

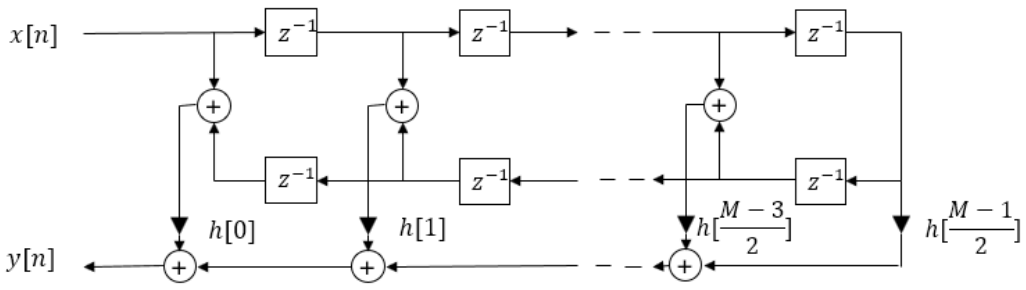

Figure 5.3.2: Symmetric FIR filter

Thus, the symmetry in a filter implies a reduction in the computational workload which can be exploited to generate efficient filter hardware and software implementations. However, the main reason for choosing a symmetric FIR filter is its linear phase response: all frequency components of the input signal are delayed in time by the same amount because the filter does not introduce phase distortion, preserving the wave shape of the signals.

LINEAR PHASE FILTER

The frequency response of a FIR filter can be expressed as follows:

$$H(e^{j\omega}) = \sum_{k=0}^{M-1} h[k] * e^{-jk\omega}$$

with h[k] being the impulse response of the filter.

A system has generalized linear phase if its frequency response can be written as:

$$H(e^{j\omega}) = A(e^{j\omega}) * e^{-j\alpha\omega + j\beta}$$

where $A(e^{j\omega})$ is a real function and α and β are constants.

A filter with generalized linear phase has a constant group delay, which is the measure of the average time delay of the filter as a function of frequency. In the definition above the group delay is α .

Four types of FIR filters with generalized linear phases exist:

1. type 1: M odd, even symmetry;
2. type 2: M even, even symmetry;
3. type 3: M odd, odd symmetry;
4. type 4: M even, odd symmetry.

Where even symmetry implies that the filter coefficients satisfy the following condition:

$h[n] = h[M-1-n]$ with n=0,1,..., (M-1)/2 if M is odd and n = 0,1,…, (N/2)-1 if M is even;

while for odd symmetry the condition is the following:

$h[n] = -h[M-1-n], 0 \leq n \leq M-1$ with n = 0,1,…, (M-1)/2 if M is odd and n = 0,1,…, (M/2)-1 if M is even.

The impulse responses of the four types of linear-phase filters are shown in Figure 5.3.3:

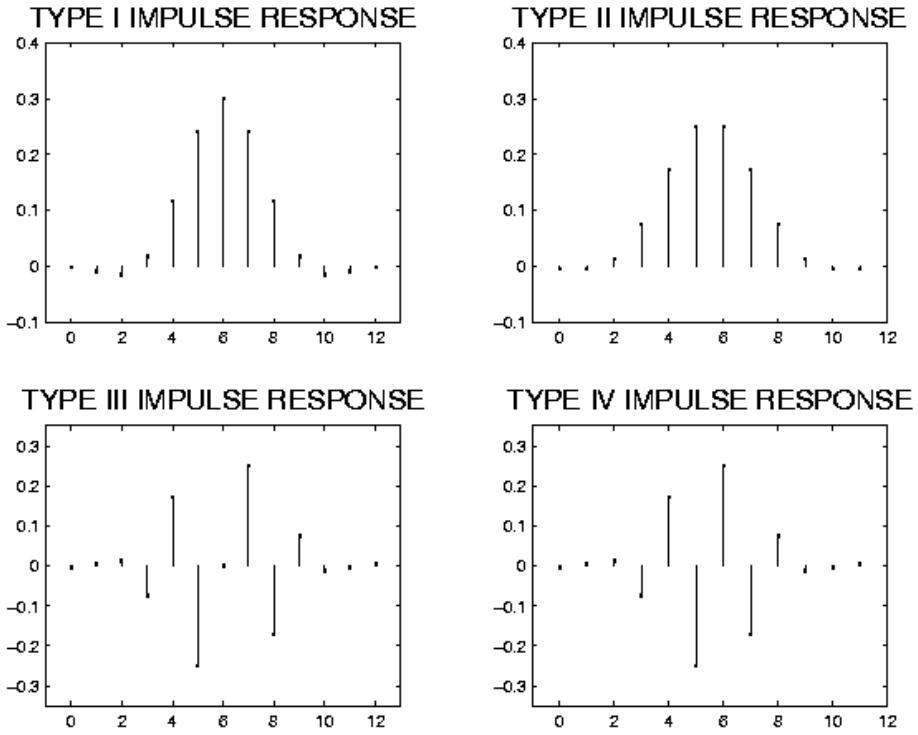

Figure 5.3.3: Impulse responses of linear-phase filters

The output y[n] of a linear-phase filter with an odd number of coefficients M can be expressed as:

$$y[n] = h[M/2] * x[n - M/2] + \sum_{k=0}^{\frac{M-1}{2}} h[k] * (x[n-k] + x[n+k-(M-1)])$$

CODE IMPLEMENTATION

Due to the memory limitations of the chosen microcontroller, we decided to implement a block filtering technique for the FIR filter.

Block filtering in signal data involves dividing the signal into blocks and applying a filtering algorithm to each block independently. This can be particularly useful in real-time processing, where the signal is too large to be processed as a whole or when the signal characteristics change over time.

To filter the input signal in blocks we need to prepend the array containing the input block to be processed an array containing the last filter-order samples of the computed output in

the previous iteration. Regarding the first iteration, the array to prepend is composed of all zeros.

The function to implement the even-symmetry FIR filter with an odd number of coefficients in C code, *simmFIR()*, receives as input parameters:

1. the array containing the filter coefficients;
2. an array DL with size equal to the filter order;
3. the array containing one block of input samples;
4. an array to store the output samples;
5. the length of the input block;
6. the length of the filter, M.

In our case, having implemented a signal generator in MATLAB, we can choose the length of the input signal and we have chosen to generate an input signal with 1991 samples which can be processed and filtered all at once. Thus, the filtering will consist of one iteration with one block corresponding to the whole input signal.

After prepending the filter-order array of zeros to the input block array, the obtained array, inputDL, is processed with the symmetric filter algorithm, which implements the equation of linear-phase filter with an odd number of coefficients M.

The steps of the algorithm are the following:

1. loop over each new input sample to compute the corresponding output sample:

   for (n = 0; n < length_input_block; n++) {

2. Initialize an accumulator variable with the middle filter coefficient multiplied by the corresponding input sample:

   *acc = coeffs[(M+1)/2] * inputDL[n + (M+1)/2];*

3. Accumulate the contributions of the remaining filter coefficients pairing the input samples according to symmetry:

```
for (k = 0; k < (M+1)/2; k++) {
    acc += (coeffs[k] * (inputDL[n + k] + inputDL[n - k + M - 1]));
}
```

4. Store the computed value in the output array:

```
output[n] = acc;
```

As the last step, the last M-1 samples of the current extended input, inpuDL, are copied to the array DL to be used in the next call; in our case the updated array DL will not be used.

5.3.2 Envelope detector implementation

The input signal to the system implemented in the microcontroller is a signal modulated in amplitude which can be expressed as:

$$x(t) = [A_0 + A_m * cos(2\pi f_m t + \theta)] * cos(2\pi f_0 t + \theta_0)$$

where:

- $A_m * cos(2\pi f_m t + \theta)$ is the modulating signal;
- $A_0 * cos(2\pi f_0 t + \theta_0)$ is the carrier signal.

AMPLITUDE MODULATION

Modulation is defined as the process of superimposing a low-frequency signal, the modulating signal which contains the information needed to be transmitted, on a high-frequency carrier signal. In particular, with amplitude modulation the amplitude of the carrier signal varies in accordance with the instantaneous amplitude of the modulating signal, Figure 5.3.4. When a carrier is amplitude-modulated by a single frequency, two sidebands are produced:

- Upper Sideband (USB): $f_0 + f_m$
- Lower Sideband (LSB): $f_0 - f_m$

The spectrum of an amplitude-modulated signal includes the carrier frequency and the two sidebands, each containing the information from the modulating signal, f_m.

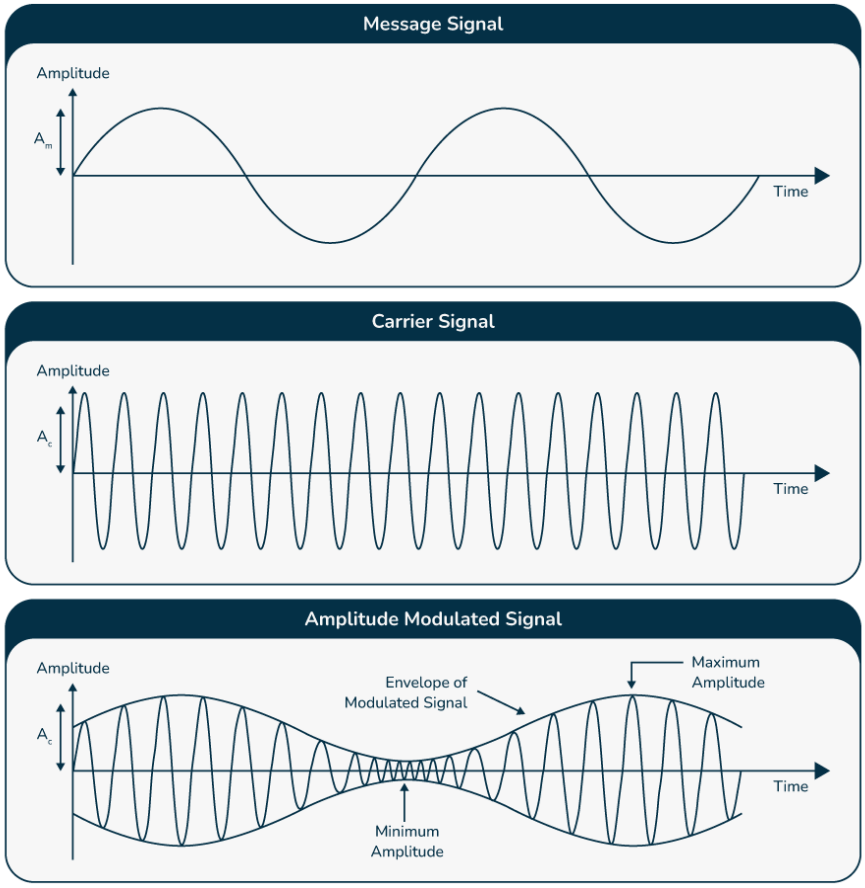

Figure 5.3.4: Amplitude modulation

AMPLITUDE DEMODULATION

To recover the wanted information of the modulating signal from the modulated signal, in our case f_m, demodulation is needed, which consists of extracting the modulating signal from the carrier signal.

There are two main techniques for amplitude demodulation:

1. synchronous detection;
2. envelope detection.

With synchronous detection, a carrier signal that is synchronized in frequency and phase with the received carrier signal is locally generated with an oscillator. Then the received

AM signal is mixed with the locally generated carrier signal. This process produces a signal that contains the original modulating signal and additional frequency components:

$$[x(t) * cos(2\pi f_0 t + \theta_0)] * cos(2\pi f_0 t + \theta_0) = x(t) * cos^2(2\pi f_0 t + \theta_0) =$$

$$\frac{1}{2}x(t) + \frac{1}{2}x(t)cos(2\pi f_0 t + 2\theta_0)$$

Thus, we obtain $\frac{1}{2}x(t)$ in baseband and the signal $\frac{1}{2}x(t)cos(2\pi f_0 t + 2\theta_0)$ modulated at frequency $2f_0$.

To isolate the original modulating signal, the output of the mixer is passed through a low-pass filter to remove the high-frequency components.

However, the need for a local oscillator that is precisely synchronized in frequency and phase with the incoming carrier wave adds complexity and is expensive. Usually, to do this a Phase-Locked Loop, PLL, is used which is a control system that adjusts the phase of the local oscillator to match the phase of the incoming carrier signal. Without a PLL the signal in baseband after mixing will be:

$\frac{1}{2}x(t)cos(\theta_0 - \varphi)$

which will oscillate as the phase θ_0 changes, giving not useful information.

Thus, given the complexity of implementing a synchronous detector an envelope detector implemented in software has been chosen for our implementation, Figure 5.3.5. With an envelope detector, we do not need to generate a carrier wave with the same phase of the incoming carrier signal for the mixing. Two sinusoidal signals, a sine and cosine signal, are generated at the frequency f_0 and then both multiplied for the modulated signal obtaining two signals which, after low-pass filtering, will be squared and sum together eliminating the sinusoidal terms with the difference of phases.

The steps are the following:

1. the modulated signal, $x(t) * cos(2\pi f_0 t + \theta_0)$, is multiplied for $cos(2\pi f_0 t + \alpha)$ and $sin(2\pi f_0 t + \alpha)$ obtaining:

$$sigMc(t) = x(t) * cos(2\pi f_0 t + \theta_0) * cos(2\pi f_0 t + \alpha)$$

$$sigMs(t) = x(t) * cos(2\pi f_0 t + \theta_0) * sin(2\pi f_0 t + \alpha)$$

2. the obtained signals, $sigMc(t)$ and $sigMs(t)$ are filtered with a low pass filter obtaining:

$$sigLc = x(t) * cos(\theta_0 - \alpha)\frac{1}{2}$$

$$sigLs = x(t) * sin(\theta_0 - \alpha)\frac{1}{2}$$

3. $sigLc$ and $sigLc$ are squared and then summed together to isolate x(t):

$$(sigLc)^2 + (sigLs)^2 = (\frac{1}{2}x(t))^2 cos^2(\theta_0 - \alpha) + (\frac{1}{2}x(t))^2 sin^2(\theta_0 - \alpha) = \left(\frac{1}{2}x(t)\right)^2$$

4. the absolute value of the square root of the previous result is taken, obtaining the rectification of the modulating signal x(t):

$$sigF(t) = \sqrt{(\frac{1}{2}x(t))^2} = \frac{1}{2}|x(t)| = \frac{1}{2}|A + (A_m cos(2\pi f_m t + \theta)|$$

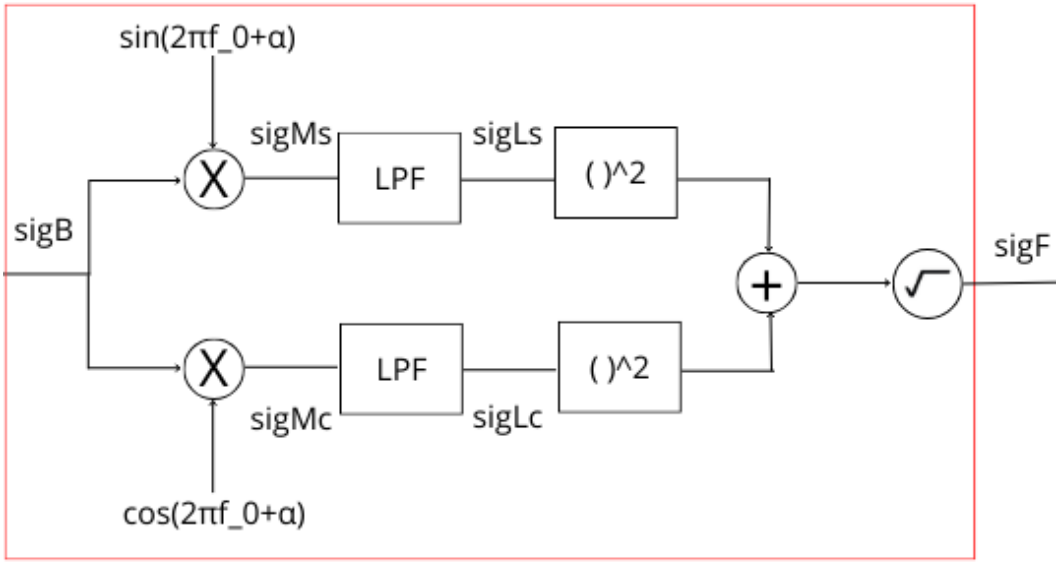

Figure 5.3.5: Implemented Envelope detector

CODE IMPLEMENTATION

To implement the envelope detector described before on the microcontroller three C functions are needed.

The *mixing()* function which takes as parameters:

1. the size of the input array;

2. the frequency f_0 of the carrier;
3. the sampling frequency fs;
4. the input signal, i.e. the modulating signal;
5. an array to store the result of the mixing between the input and the cosine signal;
6. an array to store the result of the mixing between the input and the sine signal;

The algorithm for the mixing is quite simple:

1. iterate through each sample of the input signal:

   for (i = 0; i < N; i++) {

2. for each iteration the variable omega is calculated as:

   *omega = 2 * PI * (f0 / fs) * i;*

3. a sample for each of the two carrier signals is generated for each input sample by using the floating-point cosine and sine functions, respectively cosf and sinf:

   carrier_c = cosf(omega);

   carrier_s = sinf(omega);

4. the input signal sample sigB[i] is mixed with both the sample of the cosine carrier and of the sine carrier:

   *sigMc[i] = sigB[i] * carrier_c;*

   *sigMs[i] = sigB[i] * carrier_s;*

The *arraySquare()* function which takes as input parameters:

1. the size of the input array;
2. the first array that needs to be squared and summed;
3. the second array that needs to be squared and summed;
4. the output array.

The algorithm of this function:

1. iterate through each sample of the input signal:

   for (i = 0; i < N; i++) {

2. for each input sample i the corresponding sample of the first input array and of the second input array are squared and then summed together; then the result is squared-root and stored in the output array in position i:

   *sigF[i] = sqrt(sigFc[i] * sigFc[i] + sigFs[i] * sigFs[i]);*

The *symmFIR()* function, which has already been described above in paragraph 5.3.1.

5.3.3 Fast Fourier Transform implementation

To estimate the modulating frequency f_m the Fast Fourier Transform algorithm is used which efficiently computes the Discrete Fourier Transform (DFT) and/or its inverse.

A Fast Fourier Transform (FFT) is a highly optimized implementation of the Discrete Fourier transform (DFT), which convert discrete signals from the time domain to the frequency domain. A signal is sampled over a period of time and divided into its frequency components. These components are single sinusoidal oscillations at distinct frequencies each with their own amplitude and phase. An example of this transformation is illustrated in Figure 5.3.6: over the time period measured, the signal contains three distinct dominant frequencies.

The DFT is defined by the formula:

$$X[k] = \sum_{n=0}^{N-1} x[n] * e^{-\frac{j2\pi kn}{N}}$$

where x[n] is an element of the discrete signal composed of N elements, X[k] is an element of the transformed discrete signal. The number of products needed is $O(N^2)$. Instead, a FFT algorithm allows to determine the discrete Fourier transform of an input significantly faster than computing it directly; the number of computations is reduced to $O(NlogN)$.

Popular FFT algorithms include the Cooley-Tukey algorithm, prime factor FFT algorithm, and Rader's FFT algorithm. The most used FFT algorithm is the Cooley-Tukey algorithm, which reduces a large DFT into smaller DFTs to increase computation speed and reduce complexity.

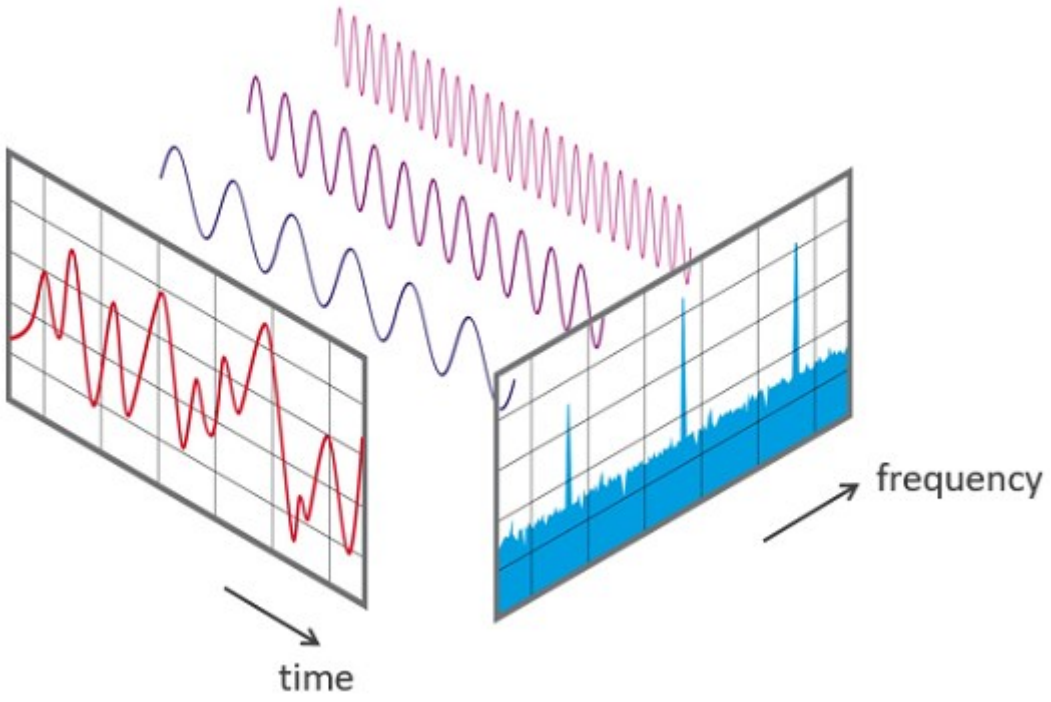

Figure 5.3.6: Example of DFT

COOLEY-TUKEY ALGORITHM

To perform the FFT we decide to implement the radix-2 Cooley-Tukey algorithm.

The Cooley-Tukey algorithm is based on the divide-and-conquer approach:

1. Divide: Split the DFT into smaller DFTs.
2. Conquer: Compute the DFTs of the smaller parts.
3. Combine: Merge the results to get the final DFT.

The most commonly used version is the radix-2 Cooley-Tukey algorithm, which works when the length of the input signal N is a power of 2; the DFT of an arbitrary composite size $N = N_1 * N_2$ is recursively broken down into many smaller DFTs of sizes N_1 and N_2, Figure 5.3.7.

CODE IMPLEMENTATION

Being a commonly used algorithm, there are several implementations in C language of the radix-2 Cooley-Tukey algorithm. The chosen implementation is reported in the Appendix.

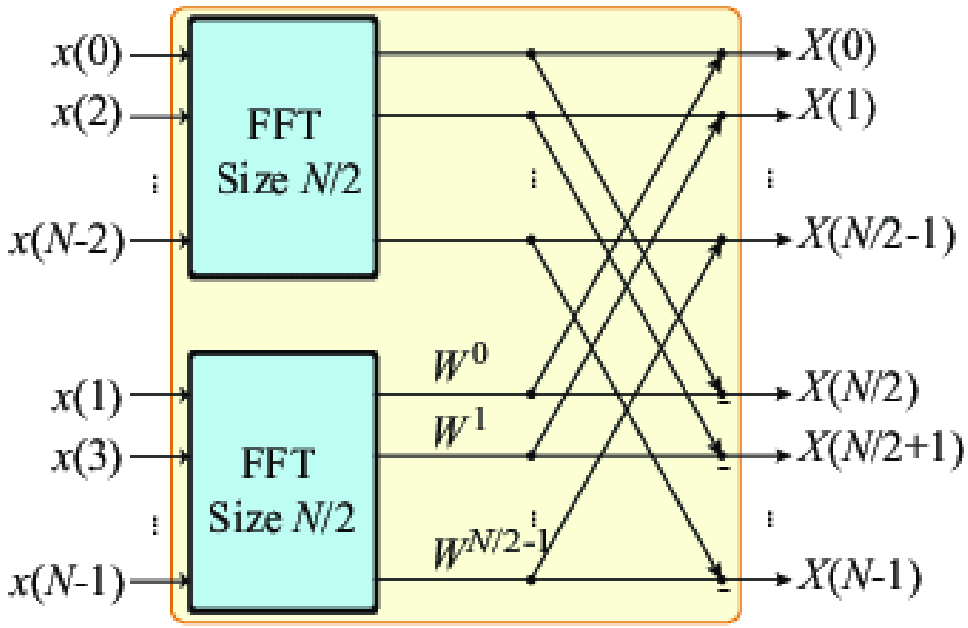

Figure 5.3.7: Illustration of Cooley-Tukey algorithm of FFT

CHAPTER 6

PERFORMANCE ANALYSIS

In this chapter, a performance analysis of fitting-based and FFT-based frequency estimation is performed.

6.1 Frequency estimation

To estimate the frequency f_m we decided to use a FFT algorithm, but another option consists of using a fitting technique. In particular, in this chapter, the results obtained with the FFT algorithm, described in Chapter 5, are compared with the ones obtained by fitting a four parameters rectified sinusoid to the signal after envelope detection.

The fitting algorithm fits a rectified sinusoidal function to a set of signal samples by optimizing four parameters: amplitude, frequency, phase, and offset. It begins by setting up multiple initial guesses for the phase parameter to increase the likelihood of finding the best fit. For each guess, it constructs a parameter vector that includes the maximum absolute value of the signal samples as the initial amplitude, a specified frequency, a phase value varying from 0 to π, and an initial offset of 0. It then optimizes these parameters to minimize the error between the actual signal and the model's output. The error is calculated by comparing the fitted signal to the original samples. The algorithm records the optimized parameters and the corresponding error for each initial guess. After all iterations, it selects the set of parameters that produced the minimum error. Finally, it generates the fitted signal using these best-fitting parameters, resulting in a rectified sinusoidal model that closely matches the original signal samples.

As can be observed from the performance curves, the fitting algorithm offers slightly better performance than the FFT-based algorithm, but requires a much larger complexity, not suitable for the low-cost solution examined in this thesis. The FFT-based algorithm was therefore selected for the final implementation.

6.2 Percentage error

The performances between the two methods are compared as percentage error defined as:

$$\%err = \frac{100 * \left| f_m - f_{m\_estimated} \right|}{f_m}$$

where f_m is the actual beat frequency, which is known since we used the simulated data for this analysis, and $f_{m\_estimated}$ is the estimated frequency.

The performances are evaluated for three parameters: the SNR ratio, the logarithm base 2 of the number of points of the FFT (M), and the number of observed sinusoid periods (NP).

6.2.1 SNR

The first parameter is the Signal To Noise ratio SNR which compares the level of the desired signal to the level of background noise.

It is typically expressed in decibels and computed as follows:

$$SNR(dB) = 10log_{10}\left(\frac{P_{signal}}{P_{noise}}\right)$$

where P_{signal} is the average power of the signal and P_{noise} is the average power of the noise.

As can be seen in Figure 6.2.1, as the SNR increases the percentage error decreases. In the case in which 512 samples are used for the FFT, for a SNR equal to -10 dB the error became acceptable, 2.5, and a threshold is reached for a SNR equal to -3 dB.

In Figure 6.2.2 a comparison between the percentage error obtained with the FFT with 512 samples and the fitting method with 5 initial phases as the SNR varies can be observed: with the fitting method, better performances are obtained. For a SNR equal to -12 dB the percentage error of the FFT method is around 6 while with the fitting method is around 3. By increasing the SNR the lowest error obtained with the FFT method is around 1.75 obtained for SNR greater than -5.5 dB while with the fitting method is 0.5 obtained for SNR greater than -1.5 dB.

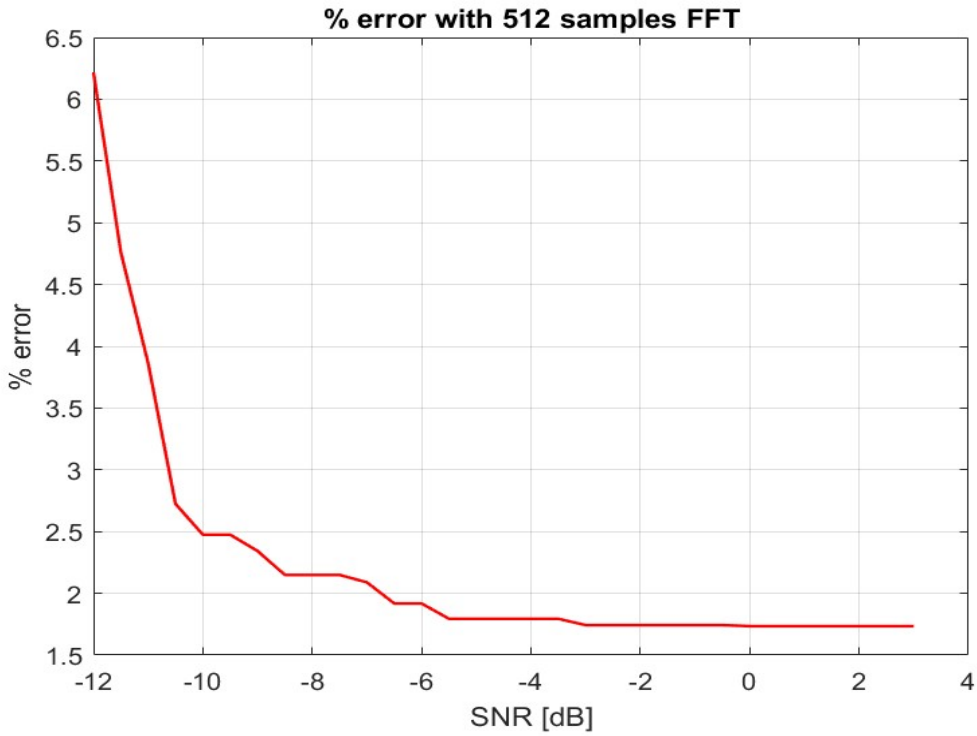

Figure 6.2.1: error vs SNR

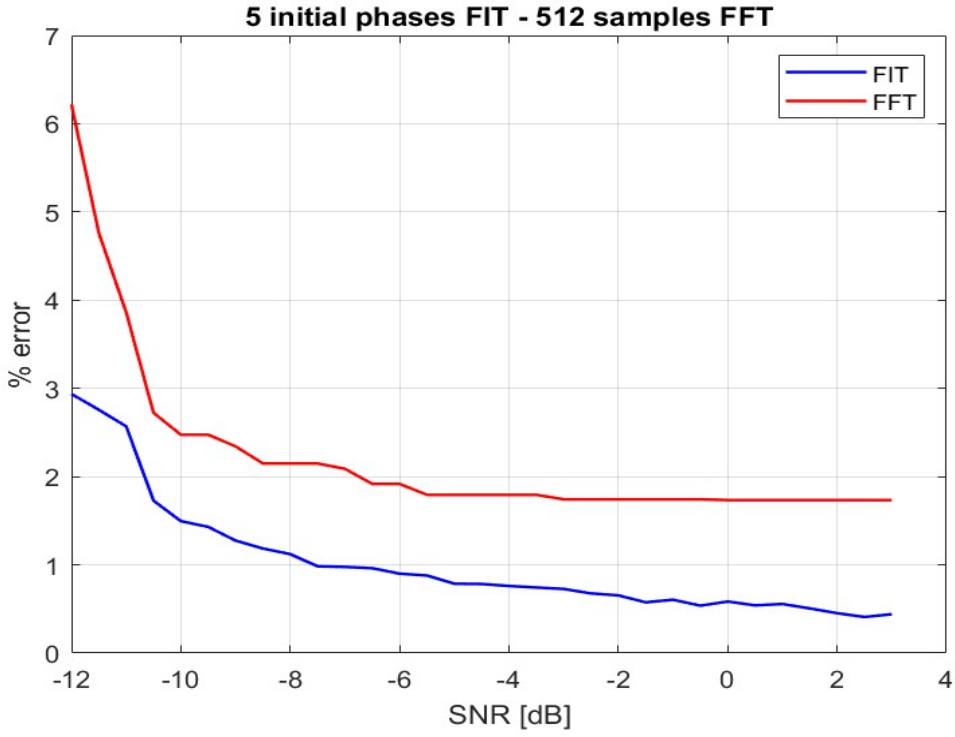

Figure 6.2.2: error vs SNR for fitting and FFT methods

6.2.2 Number of FFT points

The second parameter is logarithm base 2 of M, the number of FFT points.

In Figure 6.2.3 we can see what we obtain by varying the number of FFT points in a condition with no noise: the error decreases as the logarithm base 2 of the number of points of the FFT (M) increases and becomes steady for values of $log_2(M)$ greater than 11.

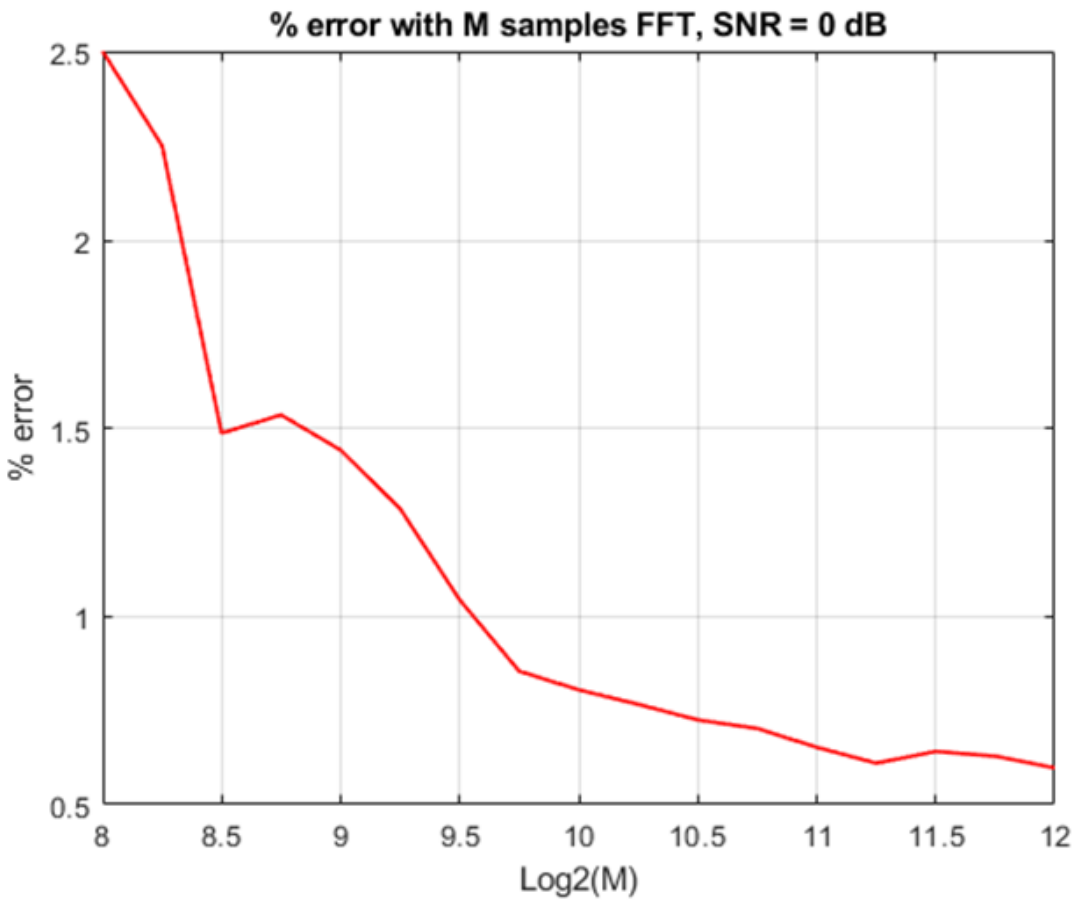

Figure 6.2.3: error vs M

The average percentage error is:

fit_m = 0.5762 for the fitting method and FFT_m = 1.0903 for the FFT method.

6.2.3 NP

The last parameter considered is the number of observed sinusoid periods (NP) which is proportional to the observation time T_{obs}, Figure 5.2.4:

$$T_{obs} = NP * T_m$$

where T_m is the period of the sinusoid. Thus, the observation time contains NP periods of the sinusoid, and N_{max} Signal samples, where $T_s$is the sampling period and $f_s = \frac{1}{T_s}$ is the sampling frequency. The link between all the timing parameters is shown in Fig. 6.2.4.

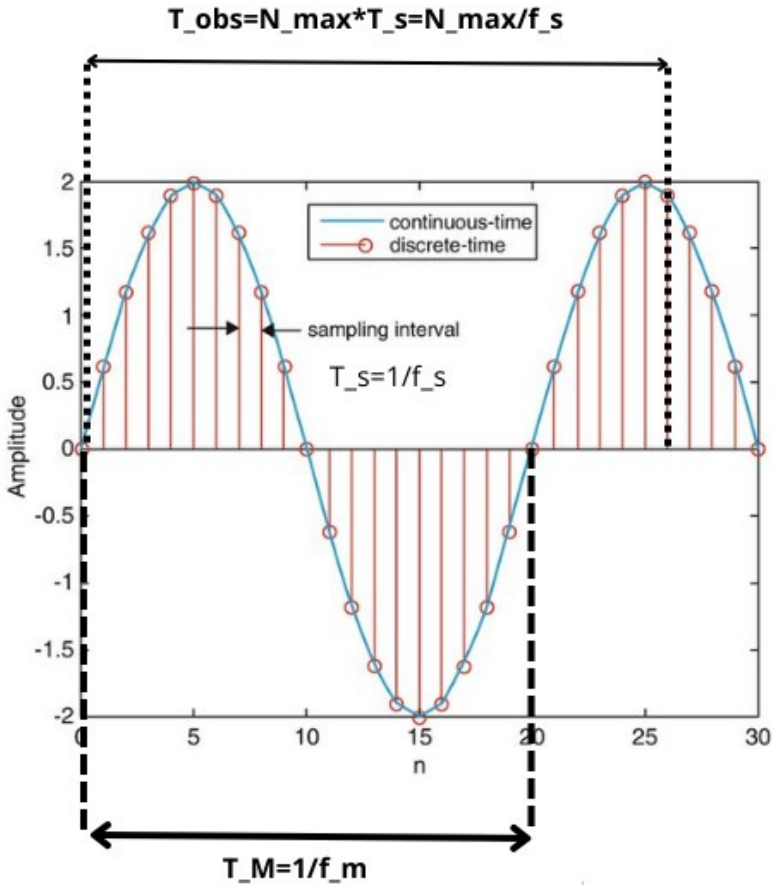

Figure 6.2.4: Observation period

As we can see in Figure 6.2.5 for M=512 and SNR=0dB the error decreases as NP increases and reaches a threshold for NP equal to 1.5 corresponding to an observation time of one-half periods of the sinusoid.

Instead, in Figure 6.2.6 a comparison between the FFT and the fitting method can be observed: the performances are similar for NP greater than 1, and for both the threshold is reached for NP greater than 1.5, but for small NP the error with the FFT method is greater than the one with the fitting method.

The average percentage error is:

fit_m = 3.2684 for the fitting method and FFT_m = 4.8690 for the FFT method.

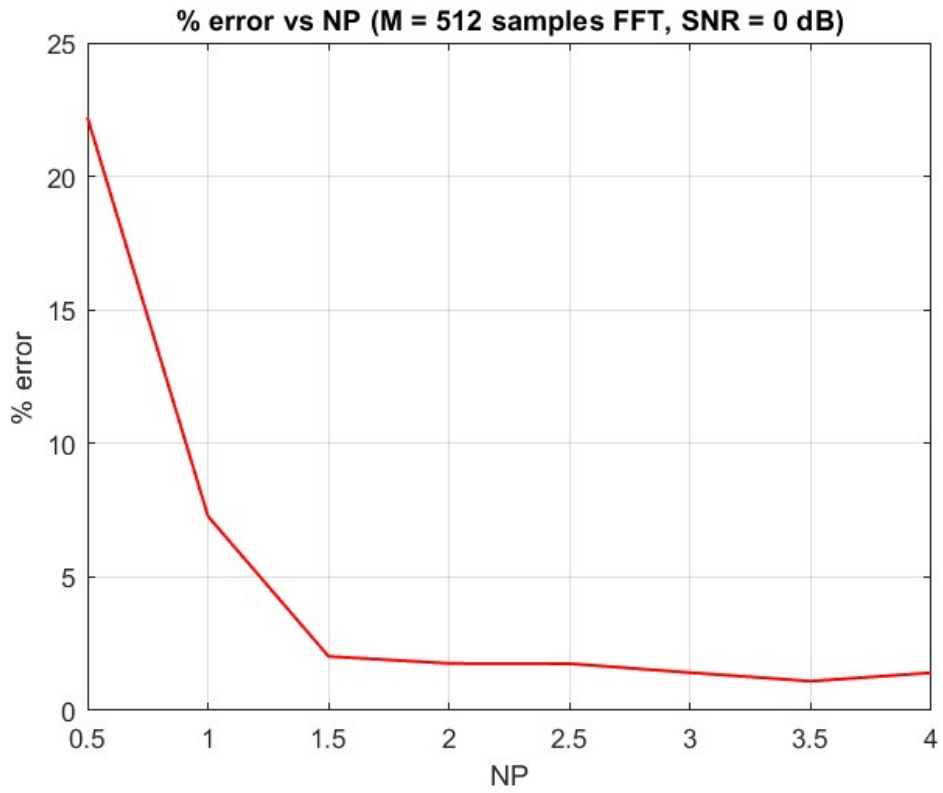

Figure 6.2.5: Error vs NP

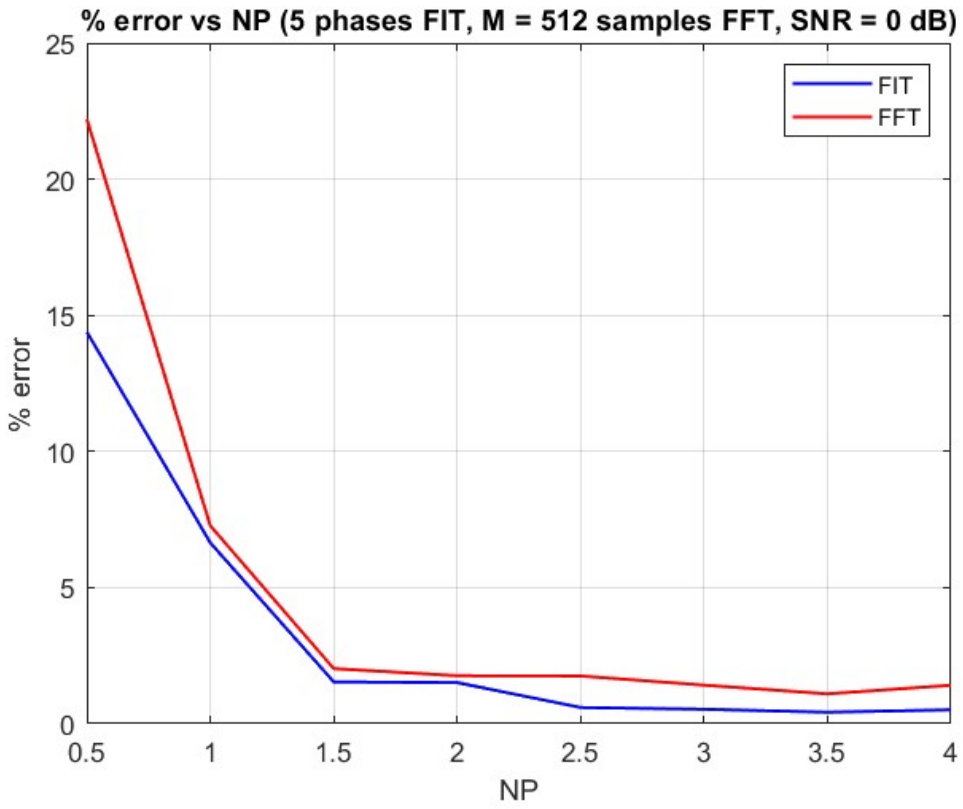

Figure 6.2.5: Error vs NP for fitting and FFT methods

CHAPTER 7

Conclusions

The objectives proposed for the thesis have been satisfied by the expected results have been obtained.

This thesis has explored the implementation of a data extraction algorithm from RADAR signals on an embedded system for precise target localization using trilateration to identify precise positions on objects in cluttered environments. Utilizing a low-complexity chirp Frequency Modulated Continuous Wave (FMCW) RADAR, the research aimed to develop a cost-effective and efficient solution for range measurement and localization.

Chapter 2 outlined the trilateration method, emphasizing its principles, geometrical and mathematical interpretations, and optimization algorithms. This method proved effective for precise localization using range information from multiple radiofrequency receivers. Chapter 3 discussed the specifics of FMCW RADAR with chirp modulation, highlighting frequency modulation patterns and the advantages of chirp modulation for improved range resolution and reduced ambiguity. In Chapter 4, the system localization scheme and schematic block diagram of the data extraction algorithm were presented, focusing on the implementation of FIR filters critical for processing RADAR signals.

Chapter 5 detailed the hardware implementation of the developed algorithm on a Texas Instruments C2000 F28379D microcontroller, covering data acquisition, flash programming, and the implementation of key components like symmetric FIR filters, the envelope detector and Fast Fourier Transform (FFT). Chapter 6 provided a performance analysis of the algorithm, examining frequency estimation through both FFT and fitting-based methods. Although the fitting algorithm offered slightly better performance, its higher complexity rendered it unsuitable for the low-cost solution targeted in this thesis. Therefore, the FFT-based algorithm, despite its marginally lower performance, was selected for the final implementation due to its lower complexity and suitability for real-time applications. The results of the performance analysis obtained varying parameters like SNR, numbers of FFT points and observation period confirm the system's effectiveness and reliability.

The thesis work confirmed the efficiency of trilateration for precise localization and the advantages of FMCW RADAR with chirp modulation for accurate range measurements at low complexity and cost. The implementation on the TI C2000 F28379D microcontroller demonstrated the feasibility of using embedded systems for complex signal processing tasks, effectively handling filtering and demodulation for real-time applications. Performance analysis highlighted critical factors affecting system accuracy, providing insights for optimizing future implementations.

References

[1] L. Zheng, *An optimization approach to indoor location problem based on received signal strength*, Master's thesis, Wright State University, 2012.

[2] An Efficient Least-Squares Trilateration Algorithm for Mobile Robot Localization, Yu Zhou, Member, IEEE

[3] W. C. Hu, W. H. Tang, —Automated least-squares adjustment of triangulation-trilateration figuresll, Journal of Surveying Engineering, 133-142, 2001.

[4] M. I. Skolnik, *Introduction to Radar Systems*, 3rd ed. New York: McGraw-Hill, 2001.

[5] M. I. Skolnik, Ed., *Radar Handbook*, 3rd ed. New York: McGraw-Hill, 2008.

[6] W. Zhong, *Linear phase FIR digital filter design using differential evolution algorithms*, Master's thesis, University of Windsor

[7] Schlichthärle, D.: *Digital Filters: Basics and Design.* Springer, Berlin, Heidelberg (2011)

[8] R. W. Hamming, *Digital Filters*. Upper Saddle River, NJ: Prentice Hall, 1999.

Appendix

FIR filter

```
void simmFIR( float *coeffs, float *DL, float *input, float *output, int length, int L )
{
   float acc=0;     // accumulator for MACs

   int n; //index for output
   int k;//index for the filter coefficients
   int M=(L-1)/2;
   printf("m is %d\n", M);

   //allocate memory for the input to the filter:
   //the first (filterLength-1) position are from the previous input (DL)
   //To which the input from main is appended (input)

   int inputDL_len= length+L-1;
   float *inputDL=malloc(inputDL_len*sizeof(float));

    //1-put in the first (filterLength -1)-positions DL: cause you have (#coeff-1) memory cell for the delay line
        memcpy( inputDL, DL,
          (L-1) * sizeof(float) );
   // 2-append the input:put the new samples at the high end of the buffer
      memcpy( &inputDL[L-1], input,
         length * sizeof(float) );
      //stampaArray(inputDL,inputDL_len);

   // apply the filter to each input sample
   for ( n = 0; n < length; n++ ) {
      // calculate output n
      acc = coeffs[M] * inputDL[n+M];
      for ( k = 0; k < M; k++ ) {
```

```
            acc += (coeffs[k]*(inputDL[n+k]+inputDL[n-k+L-1]));
        }
        output[n] = acc;
    }

    // save the last filterlegth-samples -1 of the input in DL for the next input
    memcpy( DL, &inputDL[length],(L-1) * sizeof(float) );

     free(inputDL);
}
```

Envelope detector

```
void arraySquare(int N, float *sigFc, float *sigFs, float *sigF) {
    int i;
    for (i = 0; i < N; i++) {
          sigF[i] = sqrt(sigFc[i] * sigFc[i] + sigFs[i] * sigFs[i]);
        //printf("%f\n", sigF[i]);
    }
}

    void mixing(int N, double f0_true, double fs, float *sigB, float *sigMc, float *sigMs) {
    int i;
    double omega;
    float carrier_c, carrier_s;

    for (i = 0; i < N; i++) {
        omega = 2 * PI * (f0_true / fs) * i;
        //printf("%f\n", omega);
        carrier_c = cosf(omega);
        carrier_s = sinf(omega);
        sigMc[i] = sigB[i] * carrier_c;
        sigMs[i] = sigB[i] * carrier_s;
    }
}
```

FFT

```
//FFT FUNCTIONS
typedef struct {
    double real;
    double imag;
} Complex;

// Function to perform the bit-reversal of an integer
unsigned int reverse_bits(unsigned int x, int n) {
    unsigned int result = 0;
    for (int i = 0; i < n; i++) {
        result <<= 1;
        result |= (x & 1);
        x >>= 1;
    }
    return result;
}

// Function to perform the FFT
void fft(Complex *x, int n) {
    // Bit-reversal permutation
    int log2n = log2(n);
    for (int i = 0; i < n; i++) {
        int j = reverse_bits(i, log2n);
        if (i < j) {
            Complex temp = x[i];
            x[i] = x[j];
            x[j] = temp;
        }
    }

    // Cooley-Tukey FFT
    for (int s = 1; s <= log2n; s++) {
```

```
        int m = 1 << s;
        double angle = -2.0 * M_PI / m;
        Complex wm = { cos(angle), sin(angle) };
        for (int k = 0; k < n; k += m) {
            Complex w = { 1.0, 0.0 };
            for (int j = 0; j < m / 2; j++) {
                Complex t = { w.real * x[k + j + m / 2].real - w.imag * x[k + j + m / 2].imag,
                        w.real * x[k + j + m / 2].imag + w.imag * x[k + j + m / 2].real };
                Complex u = x[k + j];
                x[k + j].real = u.real + t.real;
                x[k + j].imag = u.imag + t.imag;
                x[k + j + m / 2].real = u.real - t.real;
                x[k + j + m / 2].imag = u.imag - t.imag;
                Complex temp = { w.real * wm.real - w.imag * wm.imag, w.real * wm.imag +
w.imag * wm.real };
                w = temp;
            }
        }
    }
}
```

MATLAB code

```
%% Envelope detector and frequency estimation (with fitting and FFT)
%
% This program estimates the phase of the received signal envelope
%
% The block diagram is:
%
% signal->[decimation h0]->sigD->[BPF h1]->
% sigB->[mixing with sin and cos]->sigM->[LPF h3]->sigF->
% [selection of 1 slot]-> sigS-[fitting]->fitted
%
% sigS is a rectified sinusoid plus a constant. It is fitted with a 4
% parameters rectified sinusoid to estimate its frequency. Its dominant
% frequency (after eliminating the DC component) is also estimated using
% an FFT block and looking for the maximum value.
%
close all
clear all
warning('off','all')
warning

%% Initializations
spectra=false; timeplot=false;
mean_error_fit=[]; mean_error_FFT=[];

Nmeas=50;    % number of simulated measurements
f0_ref=200000; % reference center frequency
fs_ref=600000;   % reference sampling frewquency
Lmarg=50;   % margin for selection of starting point
fm=750;     % reference modulated signal frequency
B=f0_ref/20;    % filter bandwidth
BT=2*B;     % filter transition bandwidth
L1=100; L3=100; % the number of filter coefficients are Li +1
            % the filter delays are Li/2 (the Li values are all even)
Bmin=5*fm; % bandwidth of the low-pass filter
Nsteps=100; % Max number of optimizarion steps for each parameter
M=512;     % number of FFT points

%% Parameters of the Nmeas individual measurements
fs_i=fs_ref*ones(1,Nmeas);   % sampling frequency
f0_i=f0_ref*ones(1,Nmeas);   % center frequency
Nmax=round(2.5*fs_ref/fm)    % number of samples
N_i=Nmax*ones(1,Nmeas);      % number of acquired samples
ratio=fs_i./(3*(f0_i+B/2));  % oversampling factor
K_i=max(2.^(floor(log2(floor(ratio)))),1);  % decimation factor (power of 2)

%% Loop parameters
SNRvecdB=-12:0.5:3;
SNRvec=10.^(SNRvecdB/10);
Nloop=length(SNRvec);
% Nloop=1; % We are simulatring only one SNR value

%% Analysis for various SNRs
for indexS=1:Nloop
```

```
    dec=[]; est_freq_FFT=[]; est_amp=[]; est_freq=[]; est_pha=[]; error=[];
    rng('default');

    %% Analysis of the Nmeas measured or simulated signals
    for index=1:Nmeas
        % the index-th signal is analyzed
        f0=f0_i(index);         % center frequency
        L0=96/K_i(index);       % delay of the decimation filter
        K=K_i(index);           % decimation factor
        fs=fs_i(index);         % initial sampling frequency
        Nsig=N_i(index);        % signal contains Nsig non-zero samples

        %% Simulated signal generation
        B_s=30*fm; SNR=SNRvec(indexS); SNRdB=SNRvecdB(indexS);

[A,Am,f0_true,fm_true,pha,signal,timsig,start,stop]=fm_generator(f0,fm,B_s,Nsig,SNR,fs)
;
        fm_t(index)=fm_true;    % actual modulation frequency

        %% Spectrum visualization at the input of the demodulator(Nmax samples)
        if spectra
            spectrum(-fs/2,fs/2,fs,Nmax,signal,'Spectrum of complete signal');
            spectrum(f0_true-B/2,f0_true+B/2,fs,10*Nmax,signal,'Spectrum around f0');
        end

        %% Input signal in the demodulator
        %% No decimation (only one block is kept)
        sigD=signal(start:stop); K=1; beg(1)=start; beg(2)=stop; L0=0;
        N=length(sigD); % length of decimated signal
        fs=fs/K;        % sampling frequency after decimation

        %% Spectrum visualization after decimation (over N samplest, 1 block)
        if spectra
            spectrum(-fs/2,fs/2,fs,N,sigD,'Spectrum after decimation');
            spectrum(f0_true-B/2,f0_true+B/2,fs,10*N,sigD,'Spectrum after decimation
around f0');
        end

        %% Passband filtering
        [sigB,hPB]=passband_filtering(f0_true,B,BT,fs,L1,sigD);

        %% spectrum visualization after passband filtering over N1 samples
        % note: to visualize the spectrum with more precision we add
        % zero-valued samples with zero-padding

        if spectra
            N1=100000;
            spectrum(-fs/2,fs/2,fs,N1,sigB,'Spectrum of one block after BP filtering');
            spectrum(f0_true-B/2,f0_true+B/2,fs,N1,sigB,'Spectrum after BP filtering
around f0');
        end

        %% Mixing
        omega=2*pi*(f0_true/fs)*(0:N-1); % omega axis
        carrierc=cos(omega);        % cosine carrier
        carriers=sin(omega);        % sine carrier
        sigMc=sigB.*(carrierc');    % cosine mixing
        sigMs=sigB.*(carriers');    % sine mixing

        %% Low-pass filtering (1 stage)
```

```
        [h3]=lowpass_filtering_design2(Bmin,2*Bmin,fs,L3);
        sigFc=filter(h3,1,sigMc);    % low pass filtering
        sigFs=filter(h3,1,sigMs);    % low pass filtering
        dec=[dec floor(fs/(6*Bmin))]; % possible decimation factor

        %% Envelope detection
        sigF=sqrt(sigFc.^2+sigFs.^2);
        save("sigF.txt","sigF","-ascii");
        del=(L0+L1+L3)/2; % processing delay
        endS=min(del+beg(2),length(sigF)); % last available sample

        %% The initial delay is eliminated
        sigS=sigF(del+1:endS);

        %% spectrum visualization after envelope detection over N2 samples
        % note: to visualize the spectrum with more precision we add
        % zero-valued samples with zero-padding
        if spectra
            N2=10000;
            spectrum(-fs/2,fs/2,fs,N2,sigS,'Spectrum after demodulation');
            spectrum(-Bmin,Bmin,fs,N2,sigS,'Spectrum after demodulation (zoom)');
        end

        %% Fitting with a rectified sinusoid plus constant
        NS=length(sigS);
        save("sin.mat","NS","fs","sigS");
        [fitted,c]=fitting_procedure(sigS,NS,fm,fs,Nsteps);

        %% Estimated values
        est_amp=[est_amp abs(c(1))];
        est_freq=[est_freq c(2)];
        est_pha=[est_pha c(3)];
        % normalized error (in percentage)
        error=[error 100*sqrt(error_sin_fit4(c)/sum(abs(sigS).^2))];


        %% DC component is eliminated
        sigS1=sigS-mean(sigS);

        K2=floor(fs/(10*Bmin)); % Decimation factor
        sigS2=sigS1(1:K2:end); fs=fs/K2;
        save("sigS2.txt","sigS2","-ascii");

        %% Spectrum visualization after  DC component is eliminated over N3 samples
        if spectra
            N3=10000;
            spectrum(-fs/2,fs/2,fs,N3,sigS2,'Spectrum after demodulation (no DC)');
            spectrum(-Bmin,Bmin,fs,N3,sigS2,'Spectrum after demodulation (zoom and no
DC)');
        end

        %% Estimation of fm looking for the maximum of FFT (using M samples)
        est_freq_FFT=[est_freq_FFT freq_estimate(fs,M,sigS2)];

        %% Plot of received, envelope detected and fitted signals
        if timeplot
            % sigP=sigB(beg(1)+(L0+L1)/2+1:(L0+L1)/2+beg(2));
            sigP=sigB((L0+L1)/2+1:min((L0+L1)/2+beg(2)-beg(1)+1,end));
            figure; plot(sigP-mean(sigP),'g'); grid on; hold on;
            plot(sigS,'k'); plot(fitted,'r');plot(0*fitted,'k');
```

```
            title(['RX and DEM signals. Signal with index ' num2str(index)]);
            legend('original f0 signal','demodulated','fitted')
        end
    end

    %% Plot of the estimated values
    % figure; plot(est_pha); title('est. phase'); grid on; xlabel('signal index');
    % figure; plot(est_amp); title('est. amplitude'); grid on; xlabel('signal index');
    % figure; plot(error); title('fitting error %'); grid on; xlabel('signal index');
    % figure; plot(K_i,'-ro'); hold on; grid on; plot(ratio,'-k*');
    % legend('K_i','ratio'); title('decimation factor'); xlabel('signal index');
    mean_error_fit=[mean_error_fit mean(abs(fm_t-est_freq))];
    mean_error_FFT=[mean_error_FFT mean(abs(fm_t-est_freq_FFT))];
    SNRdB
end

%% Plot of the SNR performance
if Nloop>1
    figure
    plot(SNRvecdB,100*mean_error_fit/mean(fm_t),'b','LineWidth',1.2); hold on;
    plot(SNRvecdB,100*mean_error_FFT/mean(fm_t),'r','LineWidth',1.2); grid on
    xlabel('SNR [dB]'); title('5 initial phases FIT - 512 samples FFT');
    legend('FIT','FFT'); ylabel('% error'); xlabel('SNR [dB]');
    figure
    plot(SNRvecdB,100*mean_error_FFT/mean(fm_t),'r','LineWidth',1.2); grid on
    xlabel('SNR [dB]'); title('% error with 512 samples FFT');
    ylabel('% error'); xlabel('SNR [dB]');
end
fit_m=mean(100*mean_error_fit/mean(fm_t))
FFT_m=mean(100*mean_error_FFT/mean(fm_t))
```