Relatório de Microcontroladores e Interfaces

Trabalho realizado por: João Pedro Sá Gomes A89141
Ricardo da Silva Correia A89156

Docente: Paulo Mateus Mendes.

Universidade do Minho

Índice

17. Demonstração do funcionamento do sistema
18. Cuidados a ter durante o experimento
19. Conclusão

Introdução:

Neste trabalho, iremos converter os sinais analógicos provenientes dos eixos x, y e z de um acelerómetro para digital com o uso do microcontrolador PIC18FQ10. Seguidamente, esses dados digitais são enviados para o computador de forma a fazer o plot em tempo real dos três eixos da aceleração.Para fazer o display dos dados utilizou-se o MATLAB.

Parâmetros em conta para a realização do trabalho:

Dado que será uma pessoa a movimentar o acelerómetro, então não estamos interessados em frequências acima de 80 Hz, e de forma a evitar aliasing, utilizou-se um filtro passa-baixo à saída de cada um dos eixos do acelerômetro. Este filtro é constituído por 3 resistências em série (uma de 22k Ohm e duas de 10kOhm) e um condensador de 47 nF na qual a frequência de corte é:

$$F_c = \frac{1}{2\pi \times R \times C} = \frac{1}{2\pi \times 42k \times 47nF} \simeq 80,62\ Hz$$

Deste modo, todas as frequências acima de 80Hz vão ser filtradas/atenuadas,e por aproximação podemos assumir que a largura de banda dos 3 sinais provenientes do acelerômetro vai até aproximadamente 80 Hz.

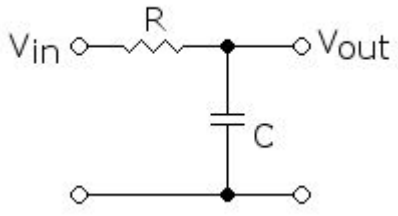


Figura 1: Esquemático de um filtro passa-baixa

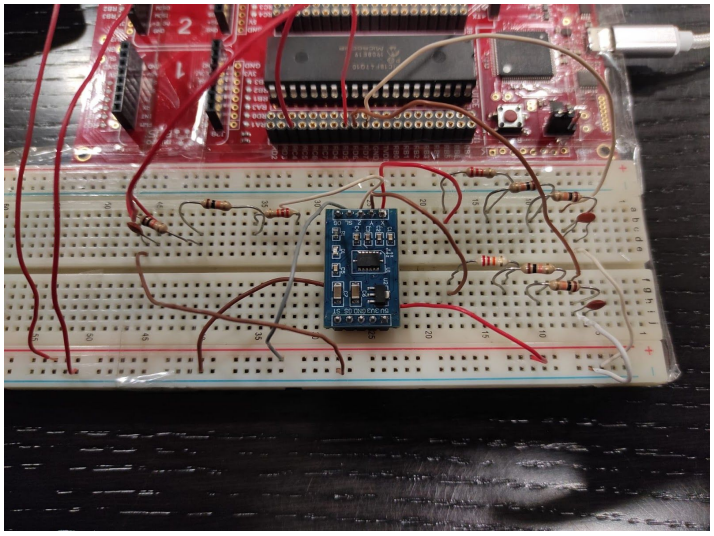


Figura 2: Montagem do filtro passa-baixo passivo no nosso projeto

Tendo estabelecido qual seria a frequência máxima de cada sinal (80 Hz),então pelo teorema de Nyquist:

$$F_a \geq 2 \times F_{máxima} \Leftrightarrow F_a \geq 2 \times 80 \Leftrightarrow F_a \geq 160\ Hz$$

Porém, como temos de fazer a multiplexagem do canal x,y e z então essa frequência tem de ser multiplicada por 3:

$$F_a \geq 480\ Hz$$

$$T_a \leq 1/480Hz \simeq 2,083\ mS$$

Ou seja, devemos converter para digital os valores de cada eixo a um período de no máximo 2,083 mS.

Outro parâmetro importante é verificar a taxa de Baud-rate necessária para conseguirmos enviar os 10 bits provenientes da conversão do ADC pela porta-série e uma vez que o PIC18F47Q10 funciona com registos de 8 bits, então somos obrigados a enviar os 10 bits do ADC em dois registos de 8 bits:
Baud-rate= $16/(2.083\ mS)$ =7681.22 bits/S
Então precisaríamos de um Baud-rate superior a 7681.22 bits/S, e o valor escolhido será dito e justificado mais à frente.

Tendo já calculado os parâmetros a ter em conta, e montando os filtros passa-baixo, então é necessário converter para digital os valores analógicos provenientes dos 3

eixos do acelerômetro, de forma a podermos manipular os dados através do computador.
Para a elaboração de toda a programação do PIC18F47Q10, utilizou-se o respectivo datasheet que explica o funcionamento do dispositivo, bem como a funcionalidade de cada registo e o material fornecido pelo professor.

Diagrama de blocos:

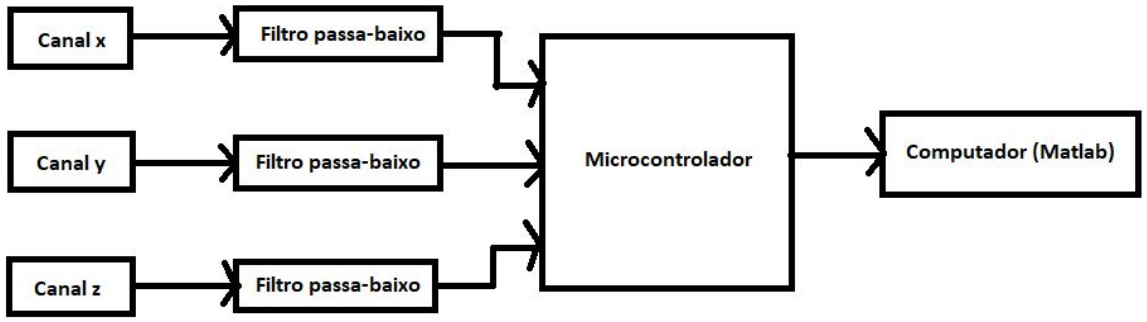


Figura 3: Diagrama de blocos do nosso trabalho

Diagrama de pinos do PIC18F47Q10:

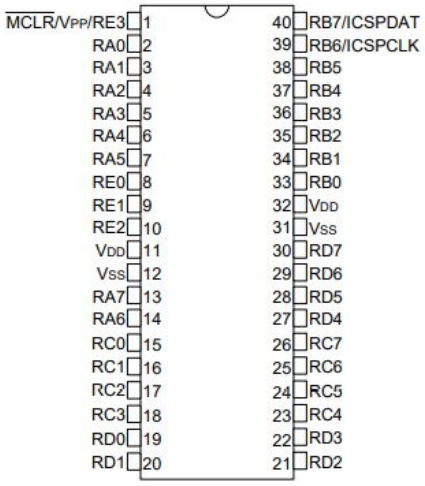


Figura 4: Diagrama de pinos do nosso microcontrolador

Ligações:

1. O ground do acelerômetro liga-se ao Ground do PIC (localizado no micro bus 2)
2. Os 3.3V do acelerômetro liga-se aos 3.3V do PIC(localizado no micro bus 2)
3. O Sleep do acelerômetro liga-se a 3.3V
4. O TX liga-se ao pino RC4
5. O eixo dos x do acelerômetro (contabilizando o filtro passa-baixo) liga-se ao pino RD7.
6. O eixo dos y do acelerômetro (contabilizando o filtro passa-baixo) liga-se ao pino RD6.
7. O eixo dos z do acelerômetro (contabilizando o filtro passa-baixo) liga-se ao pino RD5.
8. Liga-se um cabo USB do computador à porta série do PIC.

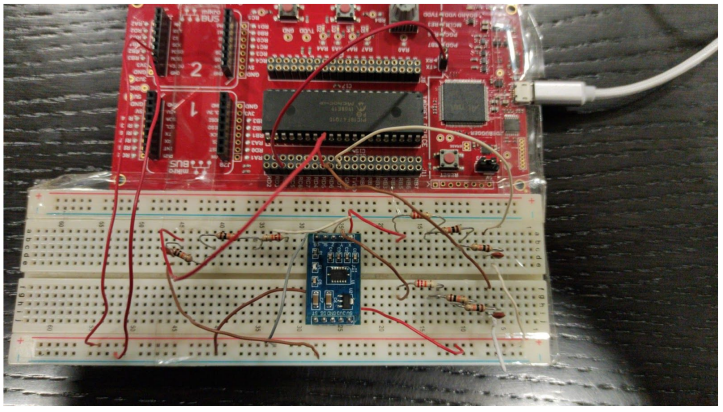


Figura 5: Ilustração das ligações realizadas

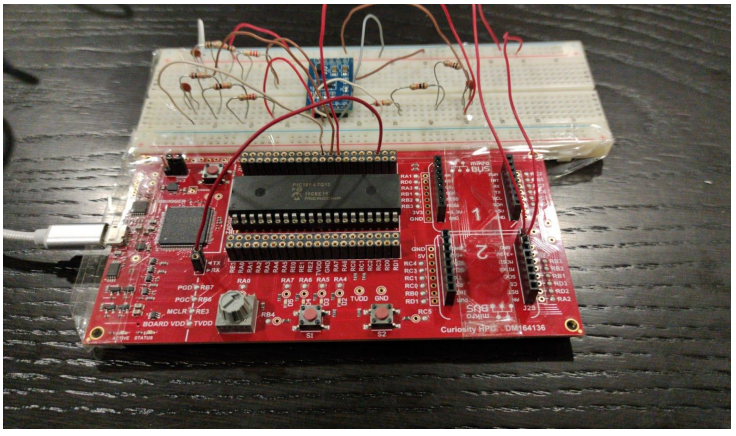


Figura 6: Ilustração das ligações realizadas

Diagrama de fluxo do código do Mplab:

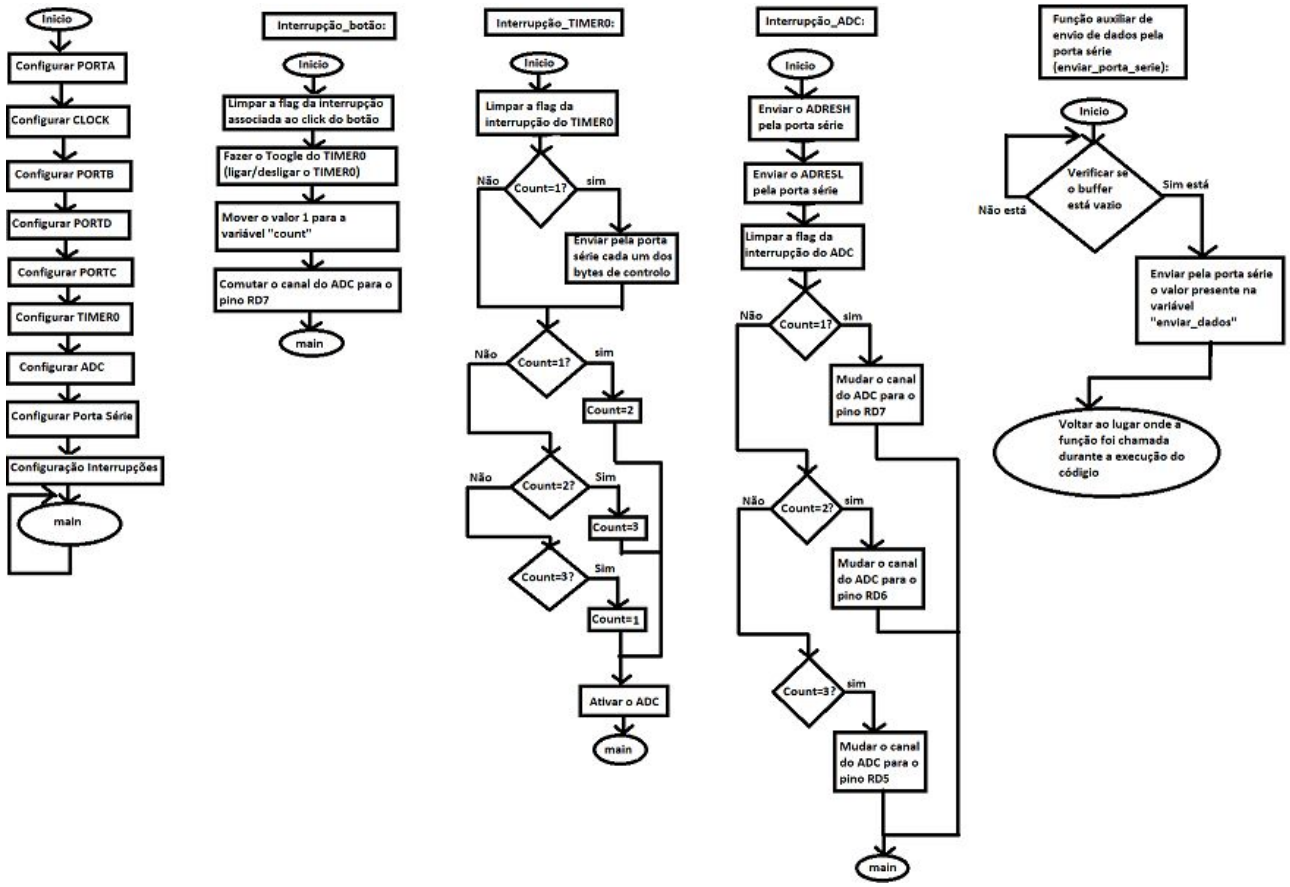


Figura 7

Resumo do funcionamento do código implementado no Mplab:

O nosso código foi projetado de forma que ao clicar no botão S1 (associado à interrupção 0), o TIMER0 (associado à interrupção do TIMER0) seja ligado ou desligado.
Caso o TIMER0 seja ligado, então ele vai contar até dar overflow e ativar a sua flag. Quando a flag do TIMER0 é ativada gera-se uma nova interrupção, e na rotina dessa interrupção (“interrupcao_TIMER0”) é verificado se o canal a converter é o do x, caso seja, são enviados os dois bytes de controlo pela porta série. Depois de verificar se é preciso enviar os bytes de controlo, é necessário manipular um contador designado “count” , que será responsável por indicar para qual canal o ADC deve comutar para

realizar a multiplexagem. No final desta mesma rotina de interrupção, ativa-se a conversão do ADC. (de notar que o TIMER0 recomeça a sua contagem)
Quando o ADC acabar de converter, então é gerada uma nova interrupção (interrupção associada ao final da conversão do ADC) e é enviado respetivamente, o byte mais significativo (ADRESH) e o byte menos significativo (ADRESL) do canal correspondente pela porta série. De seguida, verifica-se o valor que o contador "count" tem e comuta-se o canal do ADC para o pino associado de forma a realizar a multiplexagem como já se falou.
Depois isto funcionará ciclicamente até que se carregue novamente no botão S1 e se desligue o TIMER0, que posteriormente faz com que este ciclo termine.

Descrição detalhada do código implementado no Mplab:

Nota: A presença de imagens com porções de código serve para facilitar o professor, assim pode acompanhar a explicação e ver o código sem ter de mudar de janela.

- ## Configuração do PORTA

```
;=====================
;CONFIGURAÇÃO DA PORTA
;=====================
BANKSEL LATA
CLRF LATA,1 ; coloca todos os bits do LATA a zero
MOVLW 0b00000000 ; defenir todos os pinos como output
BANKSEL TRISA
MOVWF TRISA,1
MOVLW 0b00000000 ; colocar todos os pinos como digitais
BANKSEL ANSELA
MOVWF ANSELA,1
```

Figura 8

Relativamente ao porto A, apenas vamos precisar deste para acender o LED D3 (RA5). Dado o referido, inicialmente vamos limpar o registo LATA, ou seja, utilizamos o CRLF para colocar todos os bits a 0. Seguidamente, vamos querer definir se os pinos são de entrada ou saída, mas como apenas vamos querer acender os LED´s, definimos tudo como pinos de saída através do TRISA. Por fim, através do ANSELA, vamos definir todos os pinos como saídas digitais.

- **Configuração do CLOCK**

```
;====================
;CONFIGURAÇÃO DO CLOCK
;====================
BANKSEL OSCCON1
MOVLW 0b01100000  ;NOSC=110(escolhemos HFINTOSC como clock source) e NDIV(clock divider)=0000=1
MOVWF OSCCON1,1
BANKSEL OSCFRQ
MOVLW 0b00000110 ; HFFRQ 0110 -> clk= 32 MHz (32Mhz/1=clk/NDIV)
MOVWF OSCFRQ,1  ;HFINTOSC Frequency Selection Register
BANKSEL OSCEN
MOVLW 0b01000000 ;HFINTOSC ativo
MOVWF OSCEN,1
```

Figura 9

Na configuração do clock temos como objetivo obter no final um clock=32Mhz.
O bloco de oscilação interna produz fontes de relógio de alta frequência (HFINTOSC) e baixa frequência (LFINTOSC).
Nós vamos trabalhar em casos de alta frequência, ou seja, a partir do registo OSCCON1 vamos escolher o HFINTOSC (NOSC=110). No entanto, nesse mesmo registo (e através dos 4 bits menos significativos) temos a possibilidade de escolher o clock divider pretendido (NDIV=0000=1). Como escolhemos OSCFRQ=0000 0110, significa que a frequência correspondente é 32Mhz. Assim, o clock final é igual a 32Mhz/NDIV=32Mhz .

- **Configuração do PORTB**

```
;==================
;CONFIGURAÇÃO DA PORTB
;====================
BANKSEL LATB
CLRF LATB,1 ; coloca todos os bits do LATB a 0
BANKSEL TRISB
CLRF TRISB,1 ;Coloca todos os pinos como saída
BSF  TRISB,4 ;Apenas o pino RB4 vai ser um pino de entrada(botão)
BANKSEL ANSELB
CLRF ANSELB,1 ;Todos os pinos são digitais
BANKSEL INT0PPS
MOVLW 0x0C     ; 0x0C para RB4
MOVWF INT0PPS  ; liga o RB4 à INT0
```

Figura 10

Relativamente ao porto B, apenas vamos utilizar o pino RB4, ou seja, vamos estar a trabalhar unicamente com o Switch 1(S1). É este S1 que vai controlar o estado do TIMER0, ou seja, se este vai estar a ON ou OFF, como referido anteriormente.
Assim, tal como no porto A, inicialmente vamos limpar o registo LATB.
Seguidamente, através do TRISB vamos definir apenas o pino RB4 (botão) como um pino de entrada e através do ANSELB vamos definir todos os pinos como digitais.
Por fim, vamos associar o pino RB4 (S1) à interrupção 0 (registo INT0PPS).

Nota: Ver tabelas das páginas 287,288 e 289 do datasheet.

- ## Configuração do PORTD

```
;=====================
;CONFIGURAÇÃO DA PORTD
;=====================
BANKSEL LATD
CLRF LATD,1 ; coloca todos os bits do LATD a 0
BANKSEL TRISD
MOVLW 0b11100000 ; RD7,RD6,RD5 -> pinos de entrada o resto saída
MOVWF TRISD,1
BANKSEL ANSELD
MOVLW 0b11100000 ; RD7,RD6,RD5 -> analógicos o resto digital
MOVWF ANSELD,1
```

Figura 11

No PORTD, pretendemos configurar os pinos que vão estar ligados aos canais x,y e z (contabilizando o filtro passa-baixo) como input analógico, para tal fez-se o Clear do registo LATD e moveu-se o binário 0b11100000 para o registo TRISD, ou seja apenas o pinos RD7,RD6 e RD5 são inputs, de seguida moveu-se o binário 0b11100000 para o registo ANSELD, ficando somente estes 3 como analógicos.

- ## Configuração do PORTC

```
;=====================
;CONFIGURAÇÃO DA PORTC
;=====================
BANKSEL LATC
CLRF LATC,1 ; coloca todos os bits do LATC a 0
BANKSEL TRISC
; MOVLW 0b10000000 ; todos os pinos são de output exceto o RC7(é input)
MOVLW 0b00000000
MOVWF TRISC,1
BANKSEL ANSELC
CLRF ANSELC,1 ;todos os pinos são digitais
BANKSEL RC4PPS
MOVLW 0x09     ;escolhemos o EUSART1 (TX/CK)
MOVWF RC4PPS   ;colocamos o EUSART1 (TX/CK) como output no pino RC4

;Para enviar dados do PC para o PIC
;MOVLW 0b00010111  ;escolher o pino RC7 para o RX
;MOVWF RX1PPS
```

Figura 12

Na PORTC, era pretendido definir o pino RC4, à qual é ligado o TX, como output digital . Para tal colocou-se o registo TRISC todo a zero (output) e o ANSELC igualmente a zero (digital). De forma idêntica como aconteceu no PORTB, é também necessário definir o periférico de output associado ao TX (pino associado à transmissão de dados pela porta série), com o uso do datasheet verificou-se que se tinha de colocar o valor 0x09 no registo RC4PPS para o pino RC4(ver tabela da página 290 do datasheet).

- **Configuração do TIMER0:**

```
;======================
;CONFIGURAÇÃO DO TIMER0
;======================
BANKSEL T0CON0
MOVLW 0b00000001 ;1:2 postscaler(as interrupcpes sao feitas a cada 2 vezes)
                 ;timer de 8bits(bit 4)=> a contagem é apenas realizada pelo registo TMR0L
MOVWF T0CON0,1
BANKSEL T0CON1

;Tempo em que o TIMER0 incrementa: 2.048ms =1/(CLK/2)   2=postscaler

MOVLW 0b01001101   ;7,6,5 bits(010) escolhemos clock=Fosc/4
                   ;bit 4=0 => a entrada para o contador TIMER0 é sincronizada ao FOSC/4
                   ;prescaler =1101 => 8192  -> CLK=32M(FOSC)/4/8192
MOVWF T0CON1,1
BANKSEL TMR0L
CLRF TMR0L      ; contador a zero
BANKSEL TMR0H
MOVLW 0B00000001   ;TMR0H=1(vai servir para comparar o TMR0L com o TMR0H)
MOVWF TMR0H
```

Figura 13

Selecionamos um Timer de 8 bits, clock=Fosc/4, prescaler=8192, postcaler=2 , e uma vez que definimos o TIMER0 como sendo um Timer de 8 bits então é necessário definir um valor para o registo TMR0L (contador) e para o TMR0H (comparador). Para o nosso caso, definimos o TMR0L como sendo 0 e o TMR0H como sendo 1.

Cálculo:

$$T_{timer0} = 2 \times 1 \times \frac{4 \times 8192}{32Mhz} = 2,048\ mS$$

Como foi explicado no "resumo do funcionamento do código implementado no Mplab", o valor do TIMER0 vai influenciar o tempo que demora a fazer a multiplexagem de cada canal, pois a cada overflow do TIMER0, o ADC vai converter os valores do canal correspondente (x, y ou z) e estes serão enviados pela porta série. Desta forma, definiu-se um valor do TIMER0 inferior ao período de amostragem de cada canal calculado acima (2,083 mS), obedecendo ao Teorema de Nyquist.

- **Configuração do ADC:**

```
;====================
;CONFIGURAÇÃO DO ADC
;====================
BANKSEL ADPCH
MOVLW 0b00011111   ;Inicialmente
MOVWF ADPCH,1
BANKSEL ADREF
MOVLW 0b00000000  ;Vref+=Vdd(3.3V)
MOVWF ADREF,1
BANKSEL ADCLK
MOVLW 0b00001111   ;FOSC=32Mhz =>
MOVWF ADCLK,1
BANKSEL ADCON0
MOVLW 0b10000100    ;ADC=ativado(
MOVWF ADCON0,1
```

Figura 14

Em primeiro plano, vamo-nos ligar ao canal x do acelerômetro para fazer a conversão da aceleração do eixo do x quando o TIMER0 atingir o seu primeiro overflow. Assim, o nosso ADPCH terá que estar associado ao pino RD7 (ver tabela da página 542 do datasheet).
Em segundo, temos que escolher as tensões de referência, a Vref+ e a Vref-. Foi escolhido Vref+=3.3V e Vref-=0V,que por exemplo,se o valor proveniente do canal for 3.3V, o ADC converte-o para 1111111111 (aceleração positiva mais elevada,em módulo) e se for 0V, o ADC converte-o para 0000000000 (aceleração negativa mais elevada, em módulo).
Relativamente à conversão do ADC, nós queríamos que esta fosse o mais rápida possível sem que comprometesse o funcionamento do ADC, sendo assim a escolha incidiu numa frequência do ADC de 1Mhz (moveu-se o valor 0b00001111 para o registo ADCLK). Com isto, a conversão de 1 bit vai ser realizada em 1us, ou seja, um TAD=1 us (TAD mínimo possível).
Para concluir, escolhemos o right justified, ou seja, os 2 bits mais significativos estão no ADRESH e os 8 bits menos significativos no ADRESL. Estes 10 bits resultantes da conversão do ADC, vão ser todos enviados pela porta série(a partir de 2 registos) porque pretendemos trabalhar tanto nas grandes acelerações, como nas pequenas acelerações, de forma a não desperdiçar informação.

- **Configuração da Porta Série:**

```
;==========================
;CONFIGURAÇÃO DA PORTA SERIE
;==========================

MOVLW X         ;25  valor cal
BANKSEL SP1BRGL
MOVWF SP1BRGL   ; introduz-se
MOVLW 0x00
MOVWF SP1BRGH       ; aqui col
                    ; zero , e

;Assim, SP1BRG=SP1BRGH:SP1BRGL
;desired baud rate que quriamo
;Erro=(19230.769-19200/19200)*

MOVLW 0b10100000
                    ;bit 6-> 8
                    ;bit 5-> T
                    ;bit 4-> E
                    ;bit 2-> B
                    ;bit 1-> T
BANKSEL TX1STA
MOVWF TX1STA
;MOVLW 0b10010000   ; ativar a
MOVLW 0b10000000    ; ativar a
BANKSEL RC1STA
MOVWF RC1STA
MOVLW 0b00000000    ;8 bit Bau
MOVWF BAUD1CON
```

Figura 15

Embora pudéssemos escolher um Baud-Rate de 9600, dado que precisamos no mínimo 7681.22(teórico) e 7812.5=16/2.048 (prático), escolhemos um de 19200, pois precisamos de enviar os dois bytes de controlo antes mesmo de converter e enviar o valor do canal x, fazendo com que durante esse período de Timer, sejam enviados 32 bits e não só 16 bits. Devido a isso, o 9600 pode não ser suficiente, podendo originar possíveis erros no envio.
De modo a conseguirmos selecionar o Baud-Rate de 19200 foi necessário configurar os registos SP1BRGL (‘1’ por causa do EUSART1) e SP1BRGH. A partir destes escolhemos SP1BRG=25 (o SP1BRG é constituído por 2 bytes, o mais significativo é o SP1BRGH e o menos significativo é o SP1BRGL) , logo o Baud-Rate=32 Mhz/(64*(25+1))=19230.769 e o erro é:
erro=[(19230.769-19200)/19200]*100=0.16%.
Podemos ver que o erro é mínimo.
Relativamente à transmissão dos bits, vamos querer transferi-los de 8 em 8 bits.
Relativamente ao modo de transmissão de dados do EUSART, como não vamos estar sincronizados com nada, escolhemos o modo assíncrono.
Por fim, ativa-se a porta série colocando a 1 o bit 7 do RC1STA.

- **Configuração das interrupções:**

```
;====================
;ATIVAR INTERRUPÇÕES
;====================

BANKSEL PIR0
BCF PIR0, 5 ;limpa a fl
BANKSEL PIR1
BCF PIR1,0  ;limpa a fl
BANKSEL PIE0
BSF PIE0,5  ;ativa a in
BSF PIE0,0   ;ativa int
BCF PIR0,0   ;limpa a f
BANKSEL PIE1
BSF PIE1,0  ;ativa as i
BANKSEL INTCON
BSF INTCON,7  ;ativa as
BSF INTCON,6  ;ativa in
```

Figura 16

Nesta parte, temos como objetivo ativar as interrupções que serão necessárias para o nosso código e fazer o clear das respetivas flags.
As interrupções necessárias vão ser a que está associada ao ADC, ao TIMER0 e às interrupções externas (por causa do botão S1).

- **main(loop):**

```
main:

    nop
    nop
    BCF PORTA,6
    nop
    goto main
```

Figura 17

Depois da configuração dos portos, interrupções, ADC, etc… chegamos à main. Esta nada mais é que um loop infinito até que surja uma interrupção e saltemos da main para as “interrupcoes” . Quando chegamos à instrução RETFIE (presente no final da rotina da interrupção em causa) o que esta instrução nos vai fazer é retornar ao local que estávamos antes da interrupção ser acionada, ou seja, à main. Depois voltamos a “andar às voltas” na main até que surja novamente outra interrupção.

- **“interrupcoes”:**

```
interrupcoes:

    BANKSEL PIR0
    BTFSC PIR0, 5 ;flag de int
    goto interrupcao_TIMER0

    BTFSC PIR0, 0 ;ocorreu int
    goto interrupcao_botao

    BANKSEL PIR1
    BTFSC PIR1,0    ;A conversã
    goto interrupcao_ADC
```

Figura 18

Aqui as flags associadas às interrupções externas (botão S1), ADC e TIMER0 vão ser visionadas. Inicialmente observa-se se o Timer0 deu overflow. Se sim, vai-se para “interrupcao_TIMER0”, se não salta-se para a próxima verificação. Seguidamente, verifica-se se ocorreu uma interrupção externa (originada pelo botão). Caso se verifique salta-se para “interrupcao_botao”, se não salta-se mais uma vez para uma próxima instrução. Por fim, observa-se se a conversão por parte do ADC está concluída e caso a resposta seja positiva, salta-se para “interrupcao_ADC”.

- **“interrupcao_TIMER0”:**

```
interrupcao_TIMER0:; O count vai
    BANKSEL PIR0
    BCF PIR0,5 ;limpa a flag de i
    enviarcontrolo:
        MOVLW 0b00000001 ; verif
        CPFSEQ count ; se for ig
        GOTO incrementax
        MOVLW 0b11111111 ; envia
        MOVWF enviar_dados
        CALL enviar_porta_serie
        MOVLW 0b11111111 ; envia
        MOVWF enviar_dados
        CALL enviar_porta_serie
        GOTO incrementax
    incrementax:
        MOVLW 0b00000001
        CPFSEQ count ; se for ig
        GOTO incrementay
        MOVLW 0b00000010 ; o cou
        MOVWF count
        GOTO converter
    incrementay:
        MOVLW 0b00000010
        CPFSEQ count ; se for ig
        GOTO incrementaz
        MOVLW 0b00000011 ;o coun
        MOVWF count
        GOTO converter
    incrementaz:
        MOVLW 0b00000011
        CPFSEQ count ; se for ig
        nop
        MOVLW 0b00000001 ;o coun
        MOVWF count
        GOTO converter
    converter:
        BANKSEL ADCON0
        BSF ADCON0,0 ; iniciar a

    RETFIE  ;retornar da interrup
```

Figura 19

A interrupcao_TIMER0 tem 3 funcionalidades principais:

- Verificar se é necessário enviar os dois bytes de controlo.
- Determinação do “count”. Este vai indicar qual será o canal (x,y,z) que vai ser ligado ao ADC, imediatamente a seguir a cada conversão.
- Ativar a conversão do ADC.

Em primeiro plano, é necessário limpar a flag associada à interrupção do timer.
Em seguida, vamos verificar o valor do count. Se este é 1, vão ser enviados os bytes de controlo (FF) e o count passa a ter o valor 2. O count passa a ter este valor porque o canal que vamos querer conectar ao ADC a seguir ao x, é o y. Se count=2 (é o canal y que está ligado ao ADC), agora já não é necessário enviar os bytes de controlo e apenas nos basta alterar o count para 3 (a seguir ao y, vai ser o canal z que vai estar ligado ao ADC). Se o count fosse 3, este tinha que o passar para 1 para voltarmos a ter o canal x ligado ao ADC depois da conversão ser feita.

Depois, vamos iniciar a conversão do ADC.
Finalizando, utilizamos a instrução RETFIE de modo a voltarmos ao à posição de memória onde estávamos antes de ter acontecido a interrupção.

- “**interrupcao_botao”:**

```
interrupcao_botao:
    BANKSEL PIR0
    BCF PIR0, 0    ; lir
    BANKSEL PORTA
    BTG PORTA,5    ; a
    BTG T0CON0,7   ;alt

    ;as proximas  4 lir
    MOVLW 0b00000001
    MOVWF count
    MOVLW 0b00011111 ;
    BANKSEL ADPCH
    MOVWF ADPCH,1
    RETFIE  ;retornar
```

Figura 20

Objetivos desta rotina:

- Alterar o estado do TIMER0
- Sempre que se pressiona o S1 o canal x vai ser conectado ao ADC e o count passa a ser 1.

Em primeiro, tal como fizemos na rotina anterior temos de limpar as flags porque caso contrário, mal saíssemos da interrupção, esta podia ser acionada novamente.
Em segundo, o estado do TIMER0 vai ser alterado, ou seja, se este não estava a contar, passa a contar e se estava a contar, passa a não estar. Isto é necessário porque ao pressionar o botão, pela primeira vez, nós queremos que tudo comece e para isso acontecer, o TIMER0 tem de ser acionado.Por sua vez, se pressionarmos o botão pela segunda vez com o objetivo de parar ou pausar, vamos querer parar o TIMER0. Assim, este nunca vai dar overflow, o que implica que nunca se vai entrar na interrupcao_TIMER0, logo o início da conversão do ADC nunca vai acontecer e nenhum valor será enviado pela porta série.

- **“interrupcao_ADC”:**

```
interrupcao_ADC:; Nós vamos env

    BANKSEL ADRESH
    MOVFF ADRESH, enviar_dados
    CALL enviar_porta_serie ; o
    MOVFF ADRESL, enviar_dados;
    CALL enviar_porta_serie;os
    BANKSEL PIR1
    BCF PIR1,0   ;limpar a flag
    canal1:
        MOVLW 0b00000001
        CPFSEQ count ; se count
        GOTO canal2
        MOVLW 0b00011111 ; se c
        BANKSEL ADPCH
        MOVWF ADPCH,1
        GOTO fim
    canal2:
        MOVLW 0b00000010
        CPFSEQ count ; se count
        GOTO canal3
        MOVLW 0b00011110 ; se c
        BANKSEL ADPCH
        MOVWF ADPCH,1
        GOTO fim

    canal3:
        MOVLW 0b00000011
        CPFSEQ count  ;se count
        nop
        MOVLW 0b00011101 ;se co
        BANKSEL ADPCH
        MOVWF ADPCH,1
        GOTO fim

    fim:
    nop

    RETFIE  ;retornar da interr
```

Figura 21

Objetivos da “interrupcao_ADC”:

- enviar os dados obtidos na conversão
- Comutação do canal

Concluída a conversão dos valores provenientes do canal em questão, falta o envio dos dados obtidos. Dado pretendermos utilizar os 10 bits resultantes da conversão (o resultado da conversão apresenta-se nos registos ADRESH e ADRESL) e dado que só conseguimos enviar 8 bits de cada vez, teremos que os enviar (10 bits) em 2 partes. Escolheu-se, em primeiro plano, o envio dos 2 bits mais significativos (ADRESH) e depois o envio dos 8 bits menos significativos (ADRESL). O envio é realizado a partir da função “enviar_porta_serie”, sendo esta explicada mais à frente.
Na fase seguinte, lidamos com a comutação do canal através do count. Esta comutação é realizada em função do count obtido na “interrupcao_TIMER0”, ou seja, tal como referimos em cima, esta vai-nos dizer a qual canal o ADC vai estar ligado a

seguir. Se o count=1, significa que o ADC vai ser ligado ao canal x. Por sua vez, se o count=3 o ADC vai ser ligado ao canal z do acelerômetro, etc…
A conexão entre os canais e o ADC vai ser realizado a partir do registo ADPCH.
De notar que se colocou a comutação dos canais do ADC no final da rotina de interrupção "interrupcao_ADC", de forma a existir um delay considerável para a comutação entre canais tal como indicava na página 543 do datasheet (por baixo da tabela dos canais), ou seja, tal como indicam, deve existir um determinado delay entre a comutação do canal e o início da próxima conversão do ADC.

- **"enviar_porta_serie":**

```
enviar_porta_serie:

    BANKSEL PIR3
    btfss PIR3,4      ;Skip se
                      ;TX1IF=1(
                      ;TX1IF=0(
    bra enviar_porta_serie  ;
    BANKSEL TX1REG
    movff enviar_dados,TX1REG;

    ;BTG PORTA, 7
    RETURN

end
```

Figura 22

Objetivo da função "enviar_porta_serie":

- enviar o valor presente na variável "enviar_dados" para a porta série.

Inicialmente é verificado se o buffer de transmissão da porta série está vazio, caso esteja vazio é colocado o valor presente em "enviar_dados" no registo TX1REG ,que é responsável por enviar o seu valor para a porta série. Caso o buffer de transmissão esteja cheio então verifica-se ciclicamente até ele ficar vazio.

Esquema de como os dados são enviados pela porta série:

De forma a uma melhor compreensão de como os dados são enviados pela porta série, foi realizado o seguinte esquema:

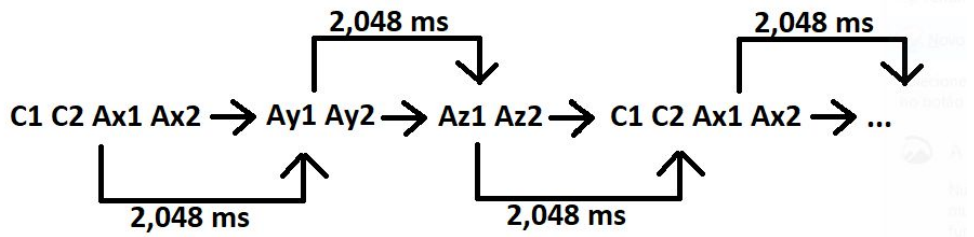


Figura 23

De notar que C1(byte mais significativo) e C2 (byte menos significativo) são constituídos por 8 bits cada um, na qual C1=11111111 e C2=11111111, que juntando resulta no valor 65535(em decimal). Quanto aos valores das acelerações, tal como já foi explicado, é enviado primeiro o byte mais significativo resultante do ADC (ADRESH) e de seguida é enviado o byte menos significativo resultante do ADC (ADRESL).

A junção adequada destes bytes faz-se no computador com o uso do MATLAB.

PC/Display

Dado concluído o envio de dados do PIC para o PC, apenas nos falta conseguir ler os valores das acelerações dos 3 eixos de modo a fazer o plot em tempo real.

Resumo do funcionamento do código do MATLAB:

Inicialmente, configurou-se a porta série com o intuito de receber os valores corretamente. De seguida criou-se uma função que tem como objetivo determinar o valor que cada eixo apresenta quando o acelerômetro está parado, para futuramente centrar os gráficos no zero. (valor de calibragem).

Caso o acelerômetro esteja parado, a tensão à saída dos canais vai estar a oscilar ligeiramente em torno de um valor entre 0V e 3.3V, e sendo assim, o valores que são convertidos não são zero mas sim um conjunto de valores bastante próximos entre 0 e 1024 (pois são utilizados os 10 bits do ADC) .
Como se pretende que os gráficos das três acelerações estejam centrados no zero, é necessário determinar o valor em que cada eixo tende a estabilizar, ou seja, quando não há aceleração. Para isso, criou-se um ciclo while que percorre um número suficiente de pontos (notar que durante este ciclo é importante que o acelerômetro esteja parado de forma a não calcular valores impróprios), de tal modo que durante

esse ciclo sejam colocados os valores de cada canal de aceleração (x,y e z) na respectiva lista. No final do ciclo while, faz-se a média dos valores de cada lista e obtém-se um valor de calibragem satisfatório para cada eixo.(De notar o facto desse valor não ser sempre igual em cada simulação, ou seja, se o sistema for ligado num determinado momento, vai ter um determinado valor de calibragem em cada eixo, porém se for desligado e depois voltado a ligar, esses valores alteram ligeiramente. Esta função consegue contornar esse problema pois a cada execução ela calcula o respectivo valor de calibragem).
De seguida elaborou-se a função capaz de realizar o plot em "tempo real". Isso será realizado através de outro ciclo while.
À medida que se vai colocando os valores da aceleração de cada eixo na respetiva lista, o ideal seria fazer o plot em "tempo real" dos valores , porém isso causava um delay significativo, inviabilizando essa estratégia. A forma que se arranjou para aproximar de um plot em "tempo real" foi guardar os valores da aceleração de cada eixo em listas voláteis (uma lista volátil para cada eixo), essas listas são voláteis pois aguardam que o seu tamanho seja de 100 pontos para que se realize o plot da aceleração dos 3 eixos, e a cada plot essas listas são limpas de forma a serem introduzidos novos valores.
Apesar de serem plotados 100 pontos de cada vez para cada um dos eixos da aceleração, o delay foi consideravelmente mais pequeno e aproximou-se mais do " tempo real" pretendido.

Diagramas de fluxo :

- **Diagrama de fluxo principal:**

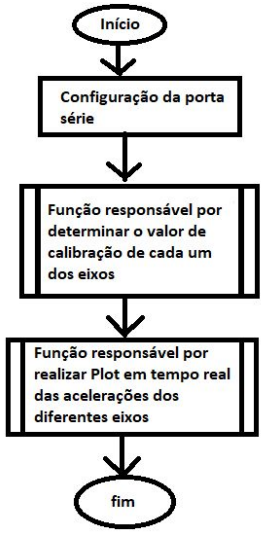


Figura 24

- **Diagrama de fluxo da função responsável por determinar o valor de calibração de cada um dos eixos:**

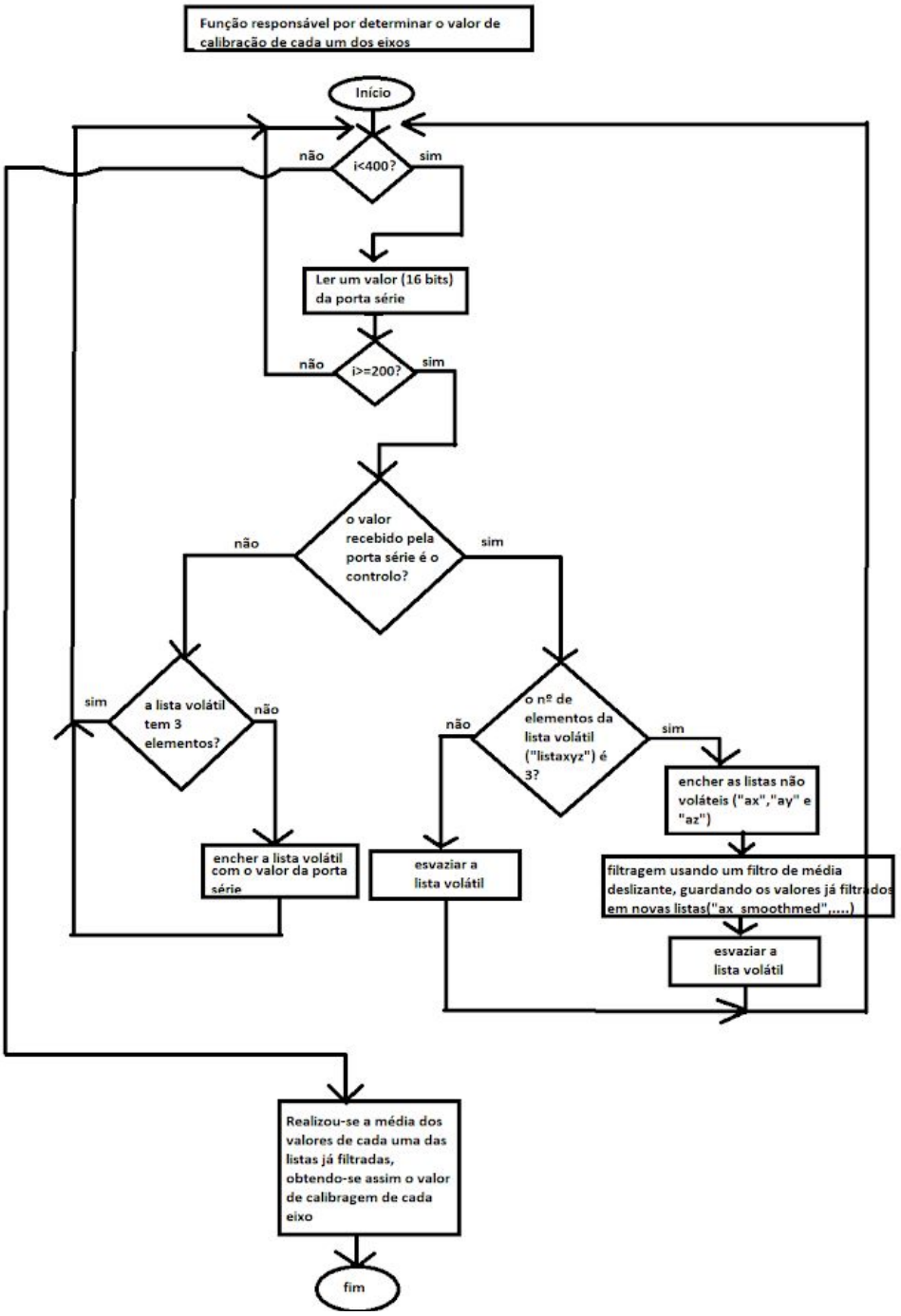


Figura 25

- **Diagrama de fluxo da função capaz de realizar o plot em "tempo real" das acelerações dos diferentes eixos:**

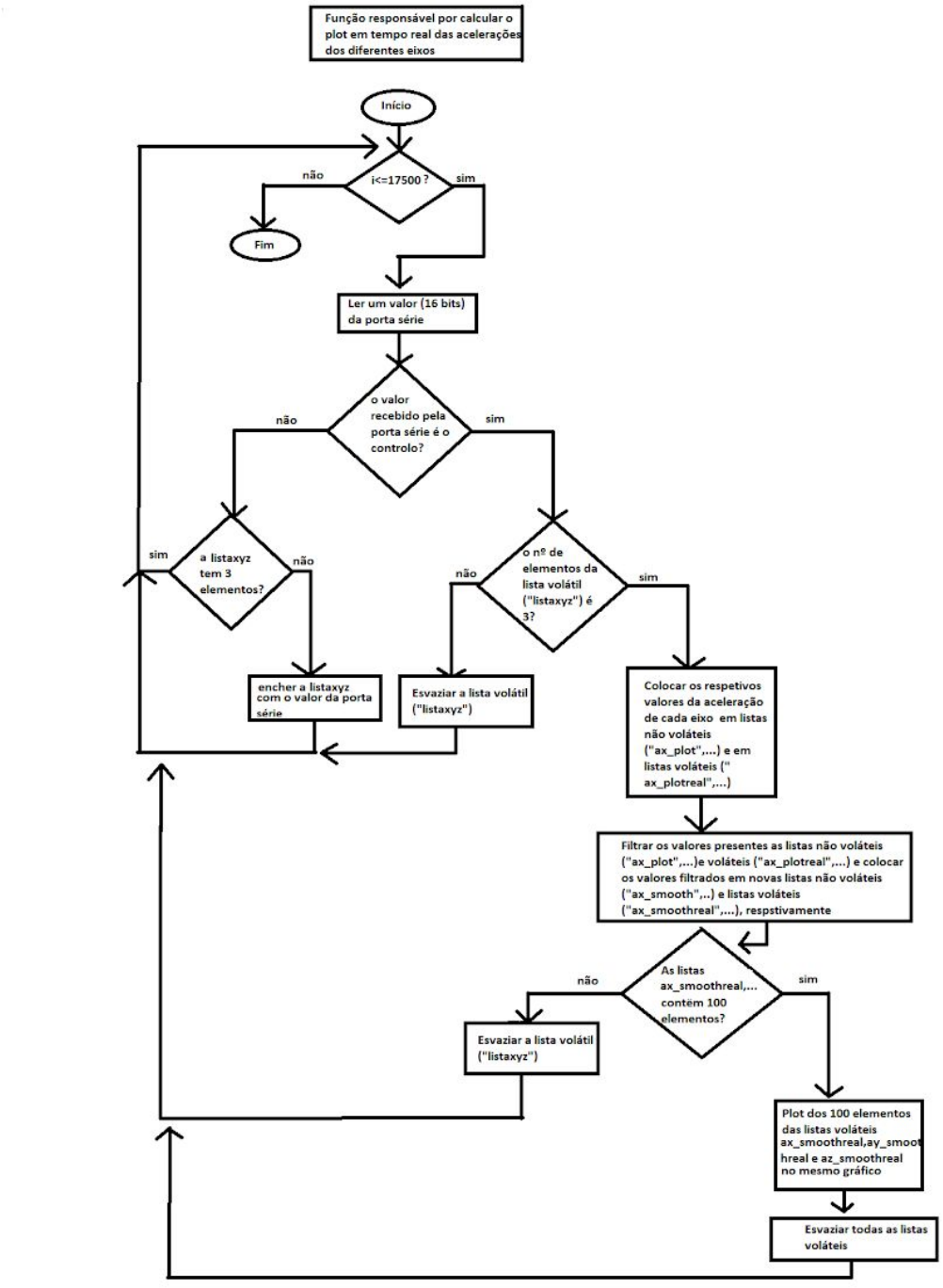


Figura 26

Nota: a implementação do código será explicado com mais detalhe a seguir

Configuração da Porta Série:

```
if ~isempty(instrfind)
   fclose(instrfind);
   delete(instrfind);
end
clear all;
close all;
clc
s = serial('COM3','BaudRate',19200);
fprintf('Pressione o botão S1 para iniciar a conversão');
set(s,'FlowControl','none');
set(s,'Parity','none');
set(s,'InputBufferSize',2);
set(s,'OutputBufferSize',1);
s.InputBufferSize=2;
s.Terminator="";
%flush(s);
s.ReadAsyncMode='continuous';
set(s,'Databits',8);
set(s,'StopBit',1);
set(s,'Timeout',10);
```

Figura 27

Configura-se a porta Série selecionando a porta a ser ligada (COM3 no nosso caso) e o Baud-Rate selecionado pelo PIC (19200).
Escolheu-se um InputBufferSize=2 de modo a guardar num array 2 bytes de uma vez (como exemplo,isto possibilita adquirir os dois bytes do eixo do x provenintes de porta série de uma vez) e um OutputBufferSize=1 (como não se enviou nada do PC para o PIC esta função não teve utilidade). Relativamente ao Databits coloca-se 8 porque a porta série envia 8 bits de cada vez.
Definiu-se a inexistência de paridade, pois não se utiliza na implementação do nosso projeto, apenas 1 Stopbit e uma leitura assíncrona contínua.
Por fim, timeout=10s que vai definir o tempo máximo de espera para concluir uma operação de escrita ou leitura.

Função responsável por determinar o valor de calibração de cada um dos eixos:

```
i=1;
ax=[]; %lista responsável por guardar os  valores da aceleração segundo x de f
ay=[]; %lista responsável por guardar os  valores da aceleração segundo y de f
az=[]; %lista responsável por guardar os  valores da aceleração segundo z de f
% listas onde se vão guardar os valores depois de passar por um filtro de
% média deslizante (smooth)
ax_smoothmed=[];
ay_smoothmed=[];
az_smoothmed=[];

controlo=0;
listaxyz=[];%lista volátil
count=1;

%Esta parte servirá para o cálculo dos valores de calibragem
%Estes valores  servirão posteriormente para deslocar
%os gráficos para o 0
%Vai ignorar os valores relacionados com falhas.

while i<400

    idn = fscanf(s); %valor recebido
    int=dec2bin(idn); %Valor em binário
    dec=bin2dec(int); %Valor em decimal
    z=(2^8)*(dec(1)) + (dec(2)); %Estabelecimento do peso de cada bit
    acel(i)=z;
    %fprintf('i=%d\n',i);
```

```
    %fprintf('i=%d\n',i);
    %fprintf('valor do acel a entrar:%d\n',acel(i));

    if i>=200       %Pontos para o cálculo da média
      if controlo==1 && acel(i)~=65535
          if count==1

              listaxyz(end+1)=acel(i); %Primeiro elemento=ax
              count=count+1;

          elseif count==2
              listaxyz(end+1)=acel(i); %Segundo elemento=ay
              count=count+1;

          elseif count==3
              listaxyz(end+1)=acel(i); %Terceiro elemento=az
              controlo=0; % zerar a variável "controlo"



          end
      end
      if acel(i)==65535 && length(listaxyz)==3  %Já tenho listaxyz=[ax,ay,az] e o va
          %fprintf('listaxyz=%d\n',listaxyz);
          ax(end+1)=listaxyz(1);      %adicionar o novo ax
          ay(end+1)=listaxyz(2);      %adicionar o novo ay
          az(end+1)=listaxyz(3);
          ax_smoothmed=smooth(ax);
```

```
          ax_smoothmed=smooth(ax);
          ay_smoothmed=smooth(ay);
          az_smoothmed=smooth(az);

          %adicionar o novo az
          %fprintf('ax=%d\n',ax);
          %fprintf('ay=%d\n',ay);
          %fprintf('az=%d\n',az);
          listaxyz=[];    % esvaziar a lista volátil
          controlo=1;
          count=1;

      elseif acel(i)==65535 && length(listaxyz) ~=3 % o valor atual é o de contro

          listaxyz=[];  % esvaziar a lista volátil
          controlo=1;
          count=1;



      end
    end


  i=i+1;
end

axmedsmooth=sum(ax_smoothmed)/length(ax_smoothmed); %média do ax depois de passar
```

```
  i=i+1;
end

axmedsmooth=sum(ax_smoothmed)/length(ax_smoothmed); %média do ax depois de passar pelo filtro sm
aymedsmooth=sum(ay_smoothmed)/length(ay_smoothmed); %média do ay depois de passar pelo filtro sm
azmedsmooth=sum(az_smoothmed)/length(az_smoothmed); %média do az depois de passar pelo filtro sm
fprintf('xmedsmooth=%d \n',axmedsmooth);

fprintf('ymedsmooth=%d\n',aymedsmooth);


fprintf('zmedsmooth=%d \n',azmedsmooth);
```

Figura 28

Inicialmente começamos por abrir a porta série e definir listas e variáveis que serão utilizadas na função.

De seguida, criou-se um ciclo while. Este ciclo while tem um limite que vai até 400, pois basta analisar um número finito de valores da porta série para conseguir calcular o valor de calibragem de cada eixo.

Na linha 48, guarda-se o array de dois bytes na variável "idn". Uma vez que o MATLAB considerava os elementos do array como caracteres ASCII foi necessário converter cada elemento do array para binário e de seguida converter de binário para decimal (linha 49 e 50). Uma vez que é preciso juntar ambos os bytes provenientes da porta série, sendo que o byte mais significativo tem mais peso que o menos significativo, então foi necessário multiplicar o decimal correspondente ao byte mais significativo por 2^8 e somar esse resultado ao decimal associado ao byte menos significativo (linha 51). Tendo toda esta conversão realizada é só guardar o valor numa lista (linha 52). De notar que a cada ciclo while, um novo valor é lido da porta série, convertido para decimal e guardado na lista.

A condição if presente na linha 56 tem como objetivo ignorar as primeiras 200 leituras provenientes da porta série, isto pois ao clicar no botão S1 são gerados picos de aceleração (devido ao facto do acelerômetro ser bastante sensível) em ambos os eixos , que iriam prejudicar o valor de calibragem dos 3 eixos.

A partir do momento em que se realiza as 200 leituras provenientes da porta série, começa-se com o algoritmo capaz de calcular esses tais valores. Este algoritmo tem a capacidade de verificar se ocorreram falhas no envio e descartar esses valores de forma a não influenciar nos nossos resultados. Por exemplo: C ax ay C ax ay az C ax ay az…. vê-se que a seguir ao primeiro controlo (C), o valor proveniente do canal z não foi enviado, porém a nossa função é capaz de ignorar esses valores e esperar pelo próximo controlo e verificar se os dados foram enviados adequadamente. Caso não haja falhas, o valor proveniente do eixo do x é guardado na lista ax, o do y na lista ay e o do z na lista az.(A explicação de como se guarda os valores na respetiva lista será falado na próxima função, uma vez que é exatamente igual).

Utilizou-se a função "smooth", esta função aplica um filtro de média deslizante aos valores armazenados nas listas "ax","ay" e "az" , sendo estes valores provenientes dos canais do acelerômetro (os valores filtrados são armazenados nas listas

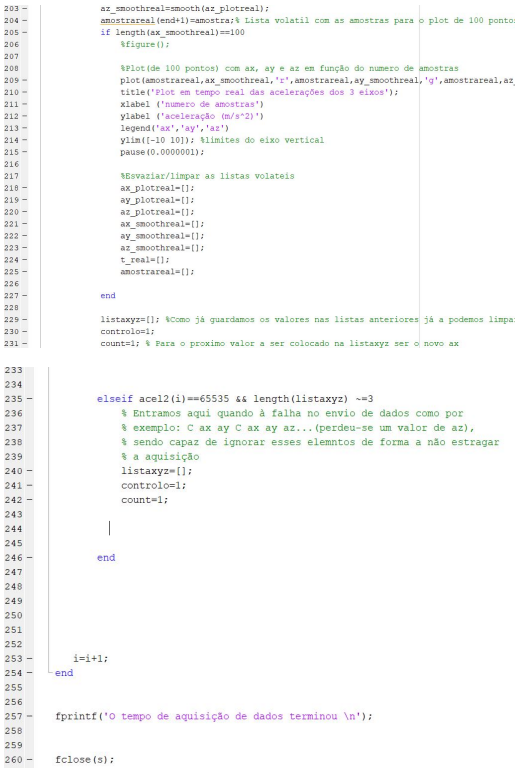
“ax_smoothmed”,“ay_smoothmed” e “az_smoothmed”). O efeito deste filtro será mostrado no plot dos gráficos, que iremos falar a seguir.
No final do ciclo while é realizada a média dos valores presentes em cada uma das listas já filtradas , guardando o resultado em “axmedsmooth”, “aymedsmooth” e “azmedsmooth”, respetivamente. Essas variáveis são os valores de calibração de cada eixo.

Função responsável por calcular o plot em tempo real das acelerações dos diferentes eixos:

```
i=1;
ax_plot=[]; %lista com os valores ax(para fazer o plot com todos os movimentos sem smooth)
ay_plot=[]; %lista com os valores ay(para fazer o plot com todos os movimentos sem smooth)
az_plot=[]; %lista com os valores az(para fazer o plot com todos os movimentos sem smooth)
ax_plotreal=[]; %lista volatil com valores do ax
ay_plotreal=[]; %lista volatil com valores do ay
az_plotreal=[]; %lista volatil com valores do az
controlo=0;
listaxyz=[];% lista volatil(vai ter 3 elementos-ax, ay, az)
count=1;
amostra=0;
listaamostra=[];% lista nao volatil com todas as amostras
amostrareal=[];% vai-nos dizer a amostra correspondente no plot(lista volatil)

% listas onde se vão guardar os valores depois de passar por um filtro de
% média deslizante (smooth)
ax_smooth=[];
ay_smooth=[];
az_smooth=[];
ax_smoothreal=[];
ay_smoothreal=[];
az_smoothreal=[];

%Servirá para o plot em "tempo real".
%Vai ignorar os valores relacionados com falhas.
%Teremos listas com todos os pontos para no final(caso queiramos) plotar,
%tento com smooth como sem smooth,todo o movimento num só gráfico
```

```
while ik<=17500 % de forma a fazer refesh do buffer apenas se vai fazer a aquisição d
    %então é só executar o programa novamente e executar novos movimentos

    idn = fscanf(s);
    int=dec2bin(idn);
    dec=bin2dec(int);
    z=(2^8)*(dec(1)) + (dec(2));
    acel2(i)=z;
    %fprintf('i=%d\n',i);
    %fprintf('valor do acel2 a entrar:%d\n',acel2(i));

    if controlo==1 && acel2(i)~=65535
          if count==1

              listaxyz(end+1)=acel2(i);
              count=count+1;

          elseif count==2
              listaxyz(end+1)=acel2(i);
              count=count+1;

          elseif count==3
              listaxyz(end+1)=acel2(i);
              controlo=0;



          end
```

```
      end
      if acel2(i)==65535 && length(listaxyz)==3

          %cada elemento destas lista é descoberto a partir da lista volatil (listaxyz=(a
          ax_plot(end+1)=(1.5*9.8*(listaxyz(1)-axmedsmooth))/((1023-axmedsmooth));
          ay_plot(end+1)=(1.5*9.8*(listaxyz(2)-aymedsmooth))/((1023-aymedsmooth));
          az_plot(end+1)=(1.5*9.8*(listaxyz(3)-azmedsmooth))/((1023-azmedsmooth));

          % aplcação de um filtro de média deslizante
          ax_smooth=smooth(ax_plot);
          ay_smooth=smooth(ay_plot);
          az_smooth=smooth(az_plot);

          amostra=amostra+1;
          listaamostra(end+1)=amostra;% lista com todas as amostras

          %Estas listas, embora volateis(vão ser limpas com um certo
          %periodo) vão ser preenchidas com os mesmos valores que as
          %listas anteriores
          ax_plotreal(end+1)=(1.5*9.8*(listaxyz(1)-axmedsmooth))/((1023-axmedsmooth));
          ay_plotreal(end+1)=(1.5*9.8*(listaxyz(2)-aymedsmooth))/((1023-aymedsmooth));
          az_plotreal(end+1)=(1.5*9.8*(listaxyz(3)-azmedsmooth))/((1023-azmedsmooth));

           % aplcação de um filtro de média deslizante
          ax_smoothreal=smooth(ax_plotreal);
          ay_smoothreal=smooth(ay_plotreal);
          az_smoothreal=smooth(az_plotreal);
```



```
            az_smoothreal=smooth(az_plotreal);
            amostrareal(end+1)=amostra;% Lista volatil com as amostras para o plot de 100 pontos
            if length(ax_smoothreal)==100
                %figure();

                %Plot(de 100 pontos) com ax, ay e az em função do numero de amostras
                plot(amostrareal,ax_smoothreal,'r',amostrareal,ay_smoothreal,'g',amostrareal,az
                title('Plot em tempo real das acelerações dos 3 eixos');
                xlabel ('numero de amostras')
                ylabel ('aceleração (m/s^2)')
                legend('ax','ay','az')
                ylim([-10 10]); %limites do eixo vertical
                pause(0.0000001);

                %Esvaziar/limpar as listas volateis
                ax_plotreal=[];
                ay_plotreal=[];
                az_plotreal=[];
                ax_smoothreal=[];
                ay_smoothreal=[];
                az_smoothreal=[];
                t_real=[];
                amostrareal=[];

            end

            listaxyz=[]; %Como já guardamos os valores nas listas anteriores já a podemos limpa
            controlo=1;
            count=1; % Para o proximo valor a ser colocado na listaxyz ser o novo ax


        elseif acel2(i)==65535 && length(listaxyz) ~=3
            % Entramos aqui quando à falha no envio de dados como por
            % exemplo: C ax ay C ax ay az...(perdeu-se um valor de az),
            % sendo capaz de ignorar esses elemntos de forma a não estragar
            % a aquisição
            listaxyz=[];
            controlo=1;
            count=1;


        end




    i=i+1;
end


fprintf('O tempo de aquisição de dados terminou \n');


fclose(s);
```

Figura 29

As primeiras linhas do ciclo while já foram explicadas anteriormente. Seguidamente temos um if que verifica se o valor da variável "controlo" é 1 e se o valor que está na lista acel2 na posição i (ou seja o valor que acabou de entrar) é diferente de 65535 (bytes de controlo). Se isto for verdade, vamos colocar na lista "listaxyz" o ax, ou ay ou az recebido (valor de cada canal do acelerômetro). Quando esta lista for constituída por 3 elementos (ax,ay e az), vamos colocá-los em listas específicas, no qual o ax_plot guarda os valores da aceleração segundo x,o ay_plot guarda o valor da aceleração segundo y e o az_plot guarda o valor da aceleração segundo z. Ao colocar estes valores nas listas correspondentes, temos de ter em atenção o valor de calibragem calculado anteriormente e a escala. Estas listas são utilizadas porque vão ser estas que nos vão permitir guardar todos os valores e consequentemente, permitir plotar um gráfico com todos os movimentos efetuados. Seguidamente, aplica-se o filtro de média deslizante de forma a "alisar" o gráfico. Os valores que foram guardados nas listas ax_plot, ay_plot e az_plot , provenientes da "listaxyz", são igualmente guardados nas respectivas listas voláteis (ax_plotreal, ay_plotreal e az_plotreal), que posteriormente são filtrados pelo filtro de média deslizante tal como as listas não voláteis acima referidas.

As listas voláteis filtradas (“ax_smoothreal”,”ay_smoothreal, e “az_smoothreal”) são as que vão ser plotadas em tempo real. Esse plot só vai acontecer quando o número de elementos das listas anteriormente referidas corresponder a 100. Depois de realizar o plot, todas as listas voláteis vão ser esvaziadas e só se volta a plotar outro gráfico quando as listas “ax_smoothreal”,”ay_smoothreal, e “az_smoothreal” voltarem a possuir 100 elementos. Isto repete-se sucessivamente.
Um “problema” desta implementação é quando se pára a conversão (pressionando no botão) e o número de elementos das listas “ax_smoothreal”,”ay_smoothreal, e “az_smoothreal” não chega a 100. Assim, como não vamos entrar no if da linha 205 não se vai plotar em tempo real estes últimos pontos. No entanto, isto não é problemático porque estes últimos valores vão ficar guardados nas listas “ax_smoothreal”,”ay_smoothreal, e “az_smoothreal” e posteriormente podemos realizar o plot destas no executável sem qualquer perda de informação.
De notar que ao mesmo tempo que se coloca os elementos da “listaxyz” nas listas anteriormente referidas, é necessário incrementar a variável “amostra”. Esta será importante porque vai ser a partir desta que vamos construir a lista “amostrareal” (nº de amostras), sendo esta última utilizada no gráficos em “tempo real”.

Através do gráfico a seguir podemos ver o efeito do filtro digital implementado:

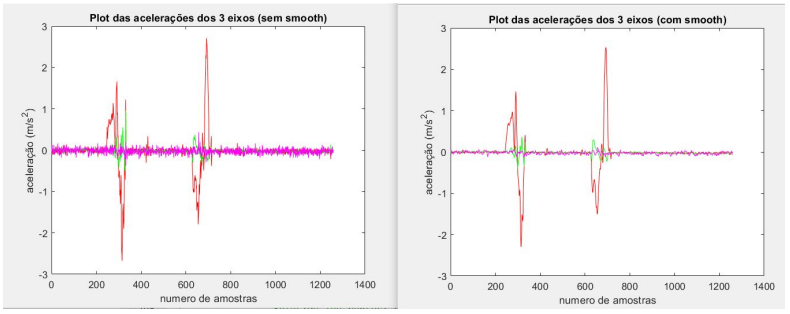


Figura 30

(Este exemplo corresponde a um movimento no sentido positivo do eixo do x, seguido por um movimento no sentido negativo do eixo do x).
Podemos ver que a filtragem “alisa” significativamente o gráfico da esquerda, permitindo uma melhor representação.

A escala é algo importante para podermos ter uma percepção mais realista do movimento efectuado.
O valor máximo da aceleração era 1.5g (retirado do datasheet do acelerômetro), logo como temos 10 bits, significa que para 1.5g temos de ter 1111111111=1023. No entanto, como estamos a retirar o respectivo valor de calibragem, o cálculo fica:

1.5*9.8 ---------- 1023-valor de calibragem
x ---------- valor - valor de calibragem

x(aceleração em m/s^2)=((valor - valor de calibragem)*1.5*9.8)/(1023-valor de calibragem)

Importante não esquecer que o valor de calibragem (“axmedsmooth”, “aymedsmooth”, “azmedsmooth”) depende de qual eixo estamos a tratar, ou seja, se tivermos a tratar do eixo dos x, temos de utilizar o “axmedsmooth” determinado acima, se for o eixo do y temos de usar o “aymedsmooth”, e se for o eixo do z temos de usar o “azmedsmooth”.

De forma a verificar se a nossa escala das acelerações estava correta, realizamos a experiência de deixar cair o acelerômetro+PIC sobre uma superfície mole (de forma a não estragar nada), enquanto estávamos a realizar o display no computador.
Fizemos 5 medidas a uma altura razoável e os valores da aceleração gravítica medidos foram: 9.30, 9.81, 9.79, 9.79 e 10.01.
Realizando a média dos 5 valores ,obtivemos aproximadamente 9.74. Este valor é próximo do valor teórico da aceleração gravítica (9.8 m/s^2), e devido a isso comprovamos a validade da nossa escala.

Alguns gráficos da experiência de deixar cair o PIC+acelerômetro:
Nota: para não sobrecarregar muito o relatório colocamos apenas dois gráficos

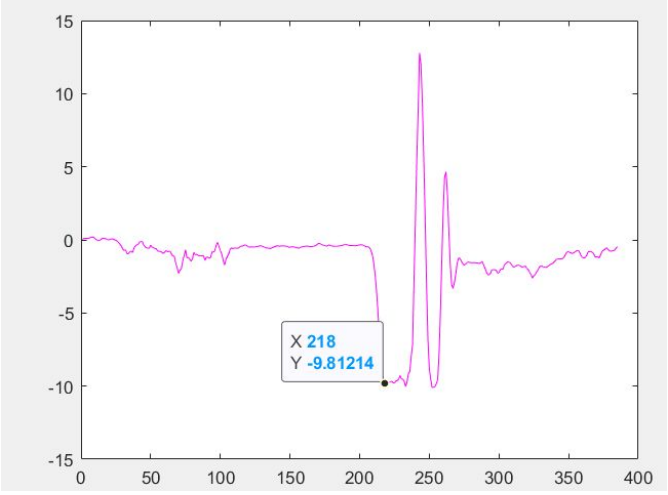


Figura 31

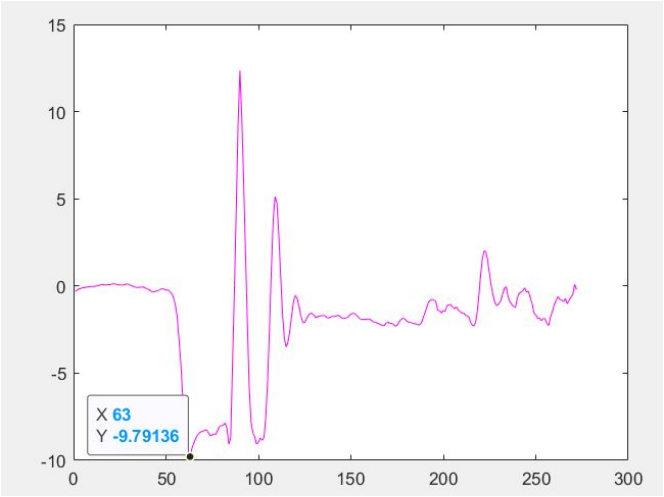


Figura 32

Demonstração do funcionamento do sistema:

1ª experiência:

Inicialmente moveu-se o PIC+acelerômetro segundo o sentido positivo do eixo do x e parou-se, logo de seguida moveu-se segundo o sentido negativo do eixo do x e voltou-se a parar, desligando-se a conversão:

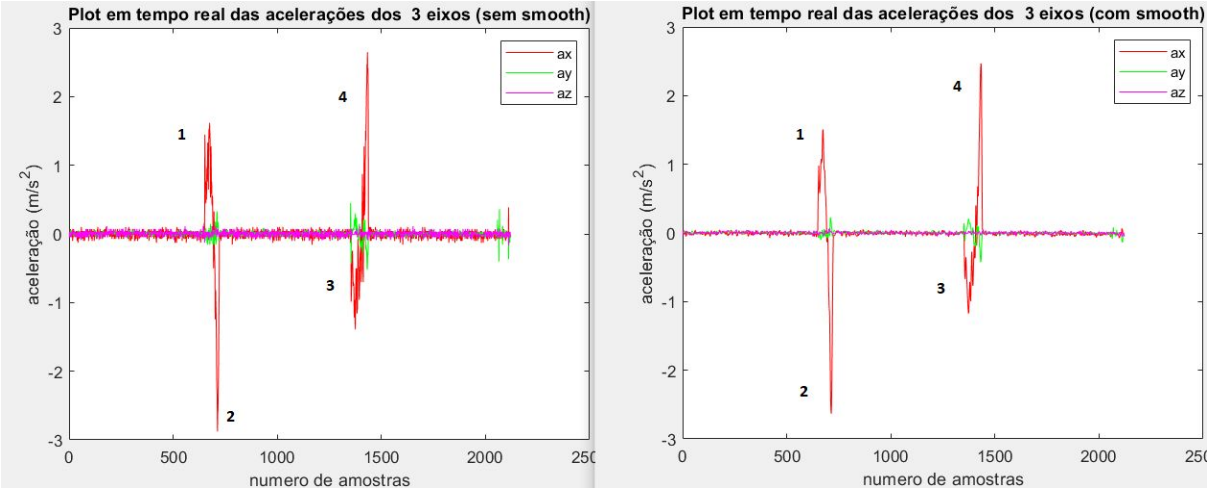


Figura 33

Podemos ver que o gráfico do ax é mais acentuado comparativamente ao dos restantes eixos, uma vez que se procurou apenas movimentar segundo o sentido de x.

Quanto ao formato do gráfico ax, podemos ver que está de acordo com o movimento efectuado, uma vez que inicialmente estava parado (aceleração zero), de seguida deslocou-se segundo x e voltou-se a parar (daí aparecer a oscilação 1(velocidade a aumentar no sentido positivo) e a 2(velocidade a diminuir no sentido positivo)). Por fim deslocou-se segundo -x e voltou-se a parar (daí aparecer a oscilação 3(velocidade a aumentar no sentido negativo) e a oscilação 4(velocidade a diminuir no sentido negativo)).

2ª experiência:

Moveu-se o PIC+acelerômetro segundo o sentido do eixo -y e parou-se, logo de seguida moveu-se segundo y e voltou-se a parar, desligando-se a conversão:

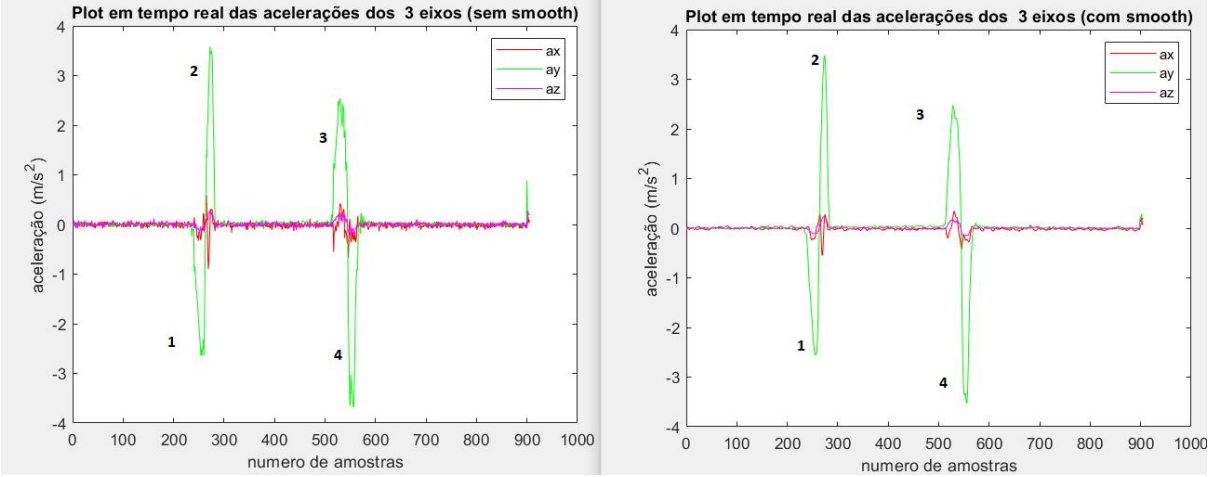


Figura 34

Podemos ver que o gráfico do ay é mais acentuado comparativamente ao dos restantes eixos, uma vez que se procurou apenas movimentar segundo o sentido de y.
Quanto ao formato do gráfico ay, podemos ver que está de acordo com o movimento efectuado, uma vez que inicialmente estava parado (aceleração zero), de seguida deslocou-se segundo -y e voltou-se a parar (daí aparecer a oscilação 1(velocidade a aumentar no sentido negativo) e a 2(velocidade a diminuir no sentido negativo)). Por fim deslocou-se segundo y e voltou-se a parar (daí aparecer a oscilação 3(velocidade a aumentar no sentido positivo) e a oscilação 4(velocidade a diminuir no sentido positivo)). Os picos que aparecem no final devem se ao clicar no botão S1 de forma a desligar a conversão, sendo que devido a isso podemos desprezar.

3ª experiência:

Moveu-se o PIC+acelerômetro segundo o sentido do eixo z e parou-se, logo de seguida moveu-se segundo -z e voltou-se a parar, desligando-se a conversão:

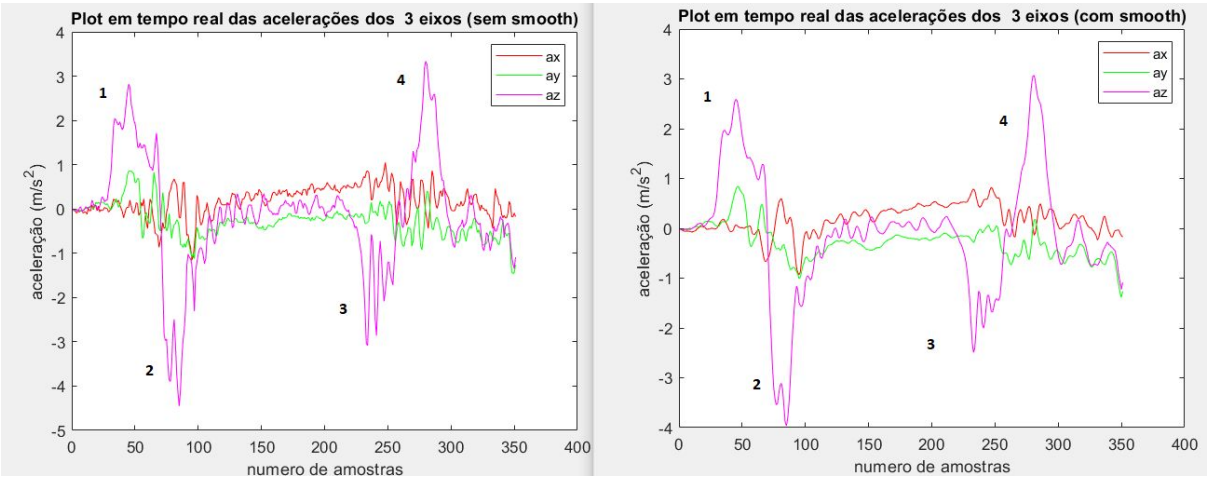


Figura 35

Podemos ver que o gráfico do az é mais acentuado comparativamente ao dos restantes eixos, uma vez que se procurou apenas movimentar segundo o sentido de z.
Quanto ao formato do gráfico ax, podemos ver que está de acordo com o movimento efectuado, uma vez que inicialmente estava parado (aceleração zero), de seguida deslocou-se segundo z e voltou-se a parar (daí aparecer a oscilação 1(velocidade a aumentar no sentido positivo) e a 2(velocidade a diminuir no sentido positivo)). Por fim deslocou-se segundo -z e voltou-se a parar (daí aparecer a oscilação 3(velocidade a aumentar no sentido negativo) e a oscilação 4(velocidade a diminuir no sentido negativo)).
Podemos verificar que o gráfico do ay e ax apresentam formatos estranhos, porém isso deve-se à dificuldade em mover o PIC+acelerômetro apenas segundo z, sem que os outros eixos sejam alterados e pelo facto de ao parar com o PIC+acelerômetro no ar, as mãos tremerem um bocado.

Cuidados a ter durante o experimento:

- Mal se comece a fazer o display , tem de se aguardar cerca de 1 segundo para o programa determinar os valores médios de cada eixo, durante esse tempo deve-se ter o cuidado de deixar o PIC+acelerômetro estável numa superfície plana.

- Assegurar que os fios estão bem conectados e ter o cuidado de durante o experimento não tocar ou abanar muito os fios, caso contrário pode haver oscilação da onda de base, sendo por isso necessário averiguar quais fios causaram isso e tentar encaixá-los melhor.

Conclusão:

A realização deste trabalho proporcionou-nos uma melhor percepção de como se faz a aquisição de sinais provenientes de um sensor e como se pode fazer o display dos mesmos através do computador, usando uma linguagem de alto nível(MATLAB). Este trabalho também nos proporcionou ver a aplicabilidade dos conhecimentos adquiridos tanto em Processamento de Sinal, como em Microprocessadores e Interfaces, bem como perceber que problemas existem quando se pretende processar um sinal proveniente de um sensor analógico e formas de os tentar corrigir.