Universidade do Minho
Escola de Engenharia

Mestrado Integrado em Engenharia de Telecomunicações e Informática

Laboratórios de Telecomunicações e Informática II

Sistema de Monitorização de Atividade Física

Fase B

Grupo 2:

David José Ressurreição Alves - A79625

José Pedro Afonso Rocha - A70020

Luís Pedro Lobo de Araújo - A73232

Guimarães, 28 de Junho de 2019

Índice

Lista de Figuras

1. Introdução

Na unidade curricular de Laboratórios de Telecomunicações e Informática II, foi-nos proposto realizar um projeto que consiste na criação de um sistema de monitorização de atividade física para doentes internados numa instituição de saúde ou de apoio social.

É de extrema importância a integração da tecnologia com os cuidados de saúde pois permite uma maior atenção aos pacientes de uma instituição de saúde e a obtenção de dados em tempo real.

Assim, maximiza-se e melhora-se as prestações de cuidados que os médicos e enfermeiros realizam obtendo, por exemplo, se não for possível estar em acompanhamento pessoal e contínuo com o doente, todos os dados relativos desse mesmo doente em qualquer lugar, sendo alertados para qualquer problema com rapidez e com a devida urgência, se necessário.

Este sistema global terá que ter um conjunto de sistemas críticos, que serão: dispositivos concentradores de dados obtidos, dispositivos sensores atuadores, servidor web e de base de dados e uma aplicação web que tenha uma interface para interacção com o utilizador. Nesta segunda fase (Fase B), o foco será o desenvolvimento do sistema Gestor de Serviço e correto funcionamento com os sistemas concentradores já implementados na fase anterior, serão ainda implementados os sistemas sensores simulados para obter uma melhor interação e testes funcionais dos sistemas Gestores de Serviço.

2. Planeamento

Nesta secção encontra-se disponível a planificação temporal, do nosso grupo para esta fase B do projeto, bem como o conjunto de ferramentas que serão utilizadas neste projeto.

2.1. Planeamento temporal

Nome	Data de início	Data de fim
FASE B	25-03-2019	29-04-2019
Compreensão de conceitos-chave	25-03-2019	25-03-2019
Introdução ao relatório da Fase B	25-03-2019	28-03-2019
Desenvolvimento da Especificação da Fase B	28-03-2019	30-03-2019
Conclusão da Especificação da Fase B	30-03-2019	31-03-2019
Entrega da Especificação da Fase B	01-04-2019	01-04-2019
Desenvolvimento de código (Gestor de Serviço)	01-04-2019	07-04-2019
Preparação de uma Base de Dados	01-04-2019	04-04-2019
Normalização da Base de Dados	01-04-2019	04-04-2019
Desenvolvimento inicial da UI do Gestor	04-04-2019	07-04-2019
Desenvolvimento de código (Concentrador)	08-04-2019	14-04-2019
Configuração protocolo TCP/IP	08-04-2019	11-04-2019
Configuração protocolo UDP	11-04-2019	14-04-2019
Criação de novos identificadores (sujeito, etc)	11-04-2019	14-04-2019
Desenvolvimento de código (Sistema Sensor Simulado)	15-04-2019	21-04-2019
Configuração protocolo UDP	15-04-2019	18-04-2019
Configuração porta UDP	15-04-2019	18-04-2019
Criação de datagramas UDP	15-04-2019	18-04-2019
Criação de ficheiros de valores sensoriais	18-04-2019	21-04-2019
Conclusão de código (Concentrador)	15-04-2019	21-04-2019
Tratamento dos dados para envio ao Gestor	15-04-2019	18-04-2019
Construção final das tramas para o Gestor	15-04-2019	21-04-2019
Conclusão de código (Gestor de Serviço)	22-04-2019	28-04-2019
Introdução de alarmes	22-04-2019	25-04-2019
Desenvolvimento final da UI do Gestor	22-04-2019	25-04-2019
Testes de comunicações com Concentrador	25-04-2019	28-04-2019
Interpretação dos dados recebidos e armazenados na BD	25-04-2019	28-04-2019
Conclusão de código (Sistema Sensor Simulado)	22-04-2019	28-04-2019
Testes de envio e recepção de dados simulados	22-04-2019	25-04-2019
Verificação de todos os ficheiros criados e gerados com valores sensoriais	25-04-2019	28-04-2019
Testes Finais	27-04-2019	29-04-2019
Conclusão do Relatório da Fase B	27-04-2019	29-04-2019
Entrega da Fase B	28-04-2019	29-04-2019

Figura 1: Diagrama com a planeamento temporal da fase B.

2.2. Ferramentas utilizadas

As ferramentas utilizadas serão as seguintes:

- Programa **GanttProject** para planeamento temporal das tarefas do grupo;
- Programa **Arduino IDE**, para editar, compilar e enviar código para a placa Arduino;
- Programa **MySQL**, para criação e gestão de bases de dados;
- Programa **GnuPlot**, para elaboração de gráficos em tempo real;
- Programa **Visual Studio Code**, para editar e compilar código;
- Programa **Visual Paradigm**, para elaboração de diagramas.
- Plataforma **Slack**, para comunicação entre os membros do grupo;
- Plataforma **GitHub**, para partilha e organização do código desenvolvido pelo grupo.
- Plataforma **Google Drive**, para partilha de ficheiros entre os membros do grupo.
- Plataforma **OverLeaf**, para elaboração de relatórios em LaTeX.
- Plataforma **Vectary**, para elaboração modelos em 3D.

3. Síntese do projeto

Na sua globalidade, este sistema irá conter um conjunto de dispositivos e sistemas críticos para o desenvolvimento deste projeto, sendo estes: sensores, servidores, base de dados, dispositivos de comunicação e microcontroladores.

Nesta fase, o grupo focou-se no desenvolvimento do Sistema Gestor de Serviço, que irá integrar a comunicação e interpretação dos dados recebidos pelos concentradores ligados à sua área.

A comunicação entre concentradores e Sistemas Gestores de Serviço é feita via TCP, sendo que os concentradores enviam os dados recebidos dos sensores para os Sistemas Gestores de Serviço, que neste caso funcionam como um servidor que guarda e interpreta toda a informação recebida.

4. Arquitetura do sistema

4.1. Esquema geral

Para esta fase B a arquitetura do nosso sistema pode ser visualizada na seguinte figura:

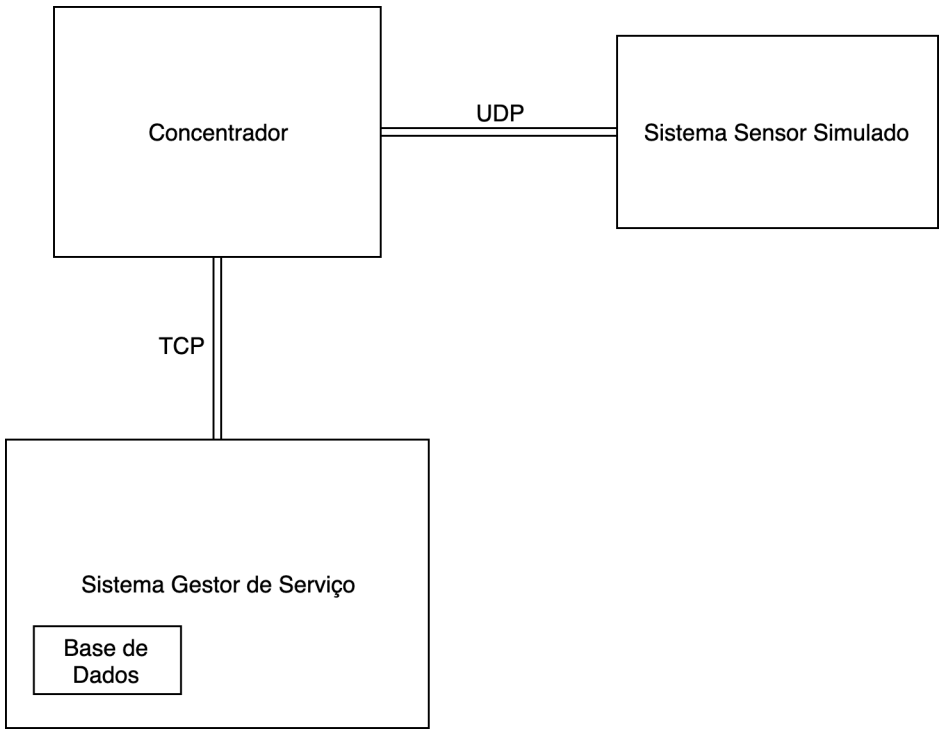

Figura 2: Arquitetura do sistema para a Fase B.

4.2. O Concentrador

Nesta fase o Concentrador vai manter a sua função de captação e normalização dos dados recebidos pelos sensores, já implementado na fase anterior. No entanto, foi adicionada a possibilidade de receber dados via UDP de um sensor simulado, o que permite uma introdução direta dos valores do sensor, que por sua vez facilita os testes e análise de resultados no Sistema Gestor do Serviço.

Além disso, foi adicionado o modulo de comunicação com o Sistema Gestor de Serviço, neste caso via TCP. Este modulo é responsável pelo envio dos dados recebidos pelo sensor para o Sistema Gestor de Serviço, na sub-secção 4.6 será descrito o protocolo de comunicação utilizado entre o concentrador e o Sistema Gestor de Serviço.

4.3. Sistema Gestor de Serviço

Para esta fase, foi criado um Sistema Gestor de Serviço que, através de dados obtidos provenientes do concentrador, faz o tratamento desses dados e recolhe-os para uma base de dados de forma que possa apresentar um conjunto de dados estatísticos sobre a monitorização da atividade física de cada sujeito internado. Este sistema possui também uma interface gráfica, onde será possível que um utilizador visualize o comportamento físico dos sujeitos na área em que este sistema está inserido. As áreas disponíveis e a lista de sujeitos monitorizados para cada Sistema Gestor de Serviço são indicados num ficheiro de configuração. A comunicação com o concentrador é implementada sobre o protocolo TCP/IP.

4.4. Sistema Sensor Simulado

De modo a poder-se realizar vários testes num ambiente em que existem vários sistemas sensores a funcionar em simultâneo para testar as funcionalidades dos Sistemas Gestores de Serviço, foi criado um Sistema Sensor Simulado.

Este sistema irá aceder a um ficheiro com valores sensoriais armazenados resultantes da monitorização do funcionamento de um sistema sensor, e comunica sobre o protocolo UDP com o Concentrador. Esta comunicação entre Concentrador e Sistema Sensor Simulado tem o mesmo protocolo de comunicação que os sensores reais, apenas altera o tipo de comunicação, uma vez que será usado UDP.

4.5. Base de Dados

De maneira a poder-se armazenar toda a informação enviada por todos os concentradores do sistema, foi criada uma Base de Dados (BD) para o efeito e de maneira a que se possa fazer o processamento de dados e apresentar um conjunto de dados estatísticos relacionados com as amostras obtidas pela monitorização da atividade física de um sujeito.

O grupo decidiu que a BD teria que ser implementada em linguagem MySQL e o planeamento inicial da mesma, a nível de entidades e relacionamentos, é demonstrado pela imagem a seguir que é um modelo conceptual de dados:

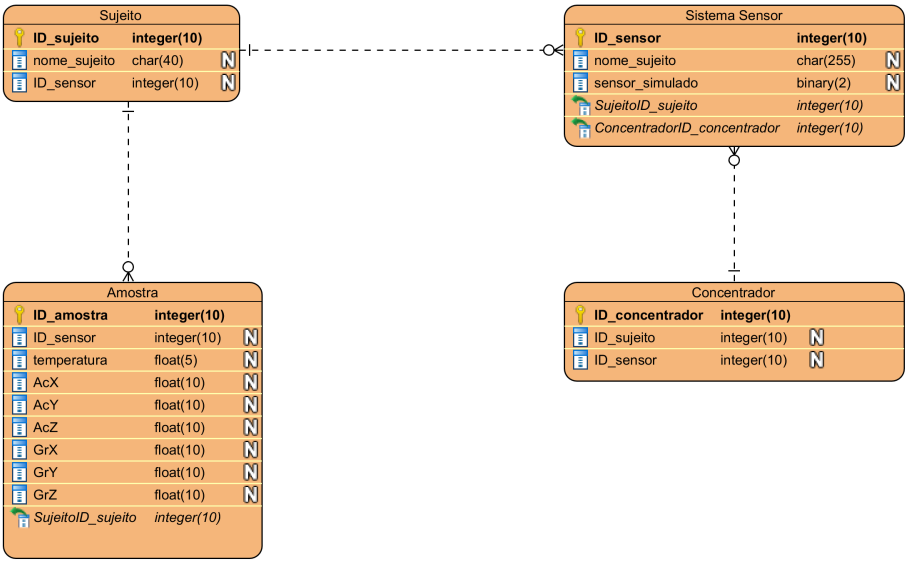

Figura 3: Modelo Conceptual de Dados da BD.

4.6. Protocolo de comunicação entre Concentradores e Gestor de Serviço

A comunicação entre concentradores e gestor de serviço é absolutamente crucial para o sistema que foi implementado, pois os dados do cliente que foram recebidos pelo concentrador terão que ser enviados para o gestor de serviço e, para isso, foi preciso um protocolo aplicacional que permita uma comunicação fiável com o correto envio e recepção de dados entre os dois serviços.

Esse protocolo é constituído por um conjunto de mensagens que são enviadas pelo concentrador, que contêm informações acerca do comportamento físico monitorizado pelos sensores atuadores e, adicionalmente, uma solução que permite que os gestores saibam quando a comunicação com um concentrador termina abruptamente. O envio destas mensagens é realizado através do protocolo TCP/IP e foram definidos os seguintes tipos de mensagem:

- **DATA** - mensagem enviada de um concentrador para um gestor de serviço com os valores das amostras recolhidas de todos os sistemas sensores que lhes estão conectados [1].

  Tal como se pode observar, definimos 1 byte para identificar o tipo de mensagem que está a ser recebida ou enviada. Neste caso para o tipo DATA, definimos o valor de 0. Os restantes campos da trama foram definidos de acordo com o tipo e tamanho dos dados que estão a ser enviados ou recebidos.

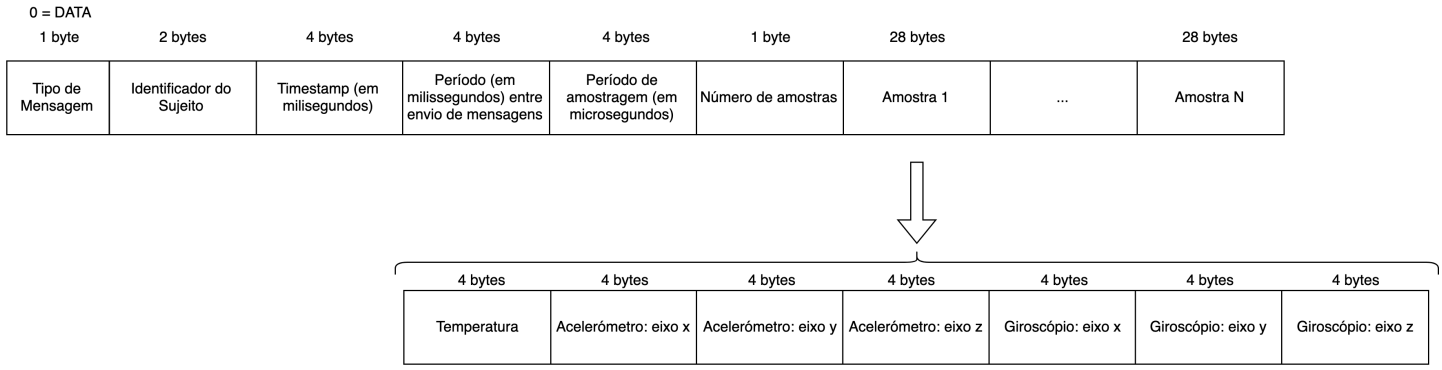

Figura 4: Definição da trama DATA.

- **ERROR** - mensagem enviada de um concentrador para um gestor de serviço a indicar um tipo de erro [1].

  Para esta trama, definimos 1 byte para identificar o tipo de mensagem que esta a ser recebida ou enviada, neste caso para o tipo ERROR, definimos o valor de 1. Os restantes campos da trama foram definidos de acordo com o tipo e tamanho dos dados que estão a ser enviados ou recebidos.

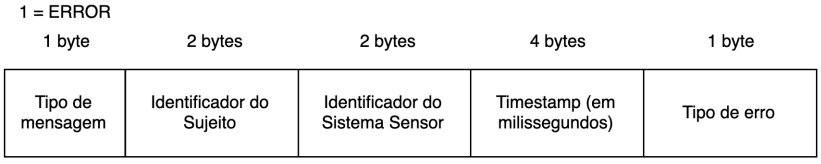

Figura 5: Definição da trama ERROR.

- **START** - mensagem enviada de um concentrador para um gestor de serviço a indicar que a recolha de amostras dum sistema sensor particular foi iniciada (ou reiniciado após uma paragem controlada) [1].

  Para esta trama, definimos 1 *byte* para identificar o tipo de mensagem que esta a ser recebida ou enviada, neste caso para o tipo START, definimos o valor de 2. Os restantes campos da trama foram definidos de acordo com o tipo e tamanho dos dados que estão a ser enviados ou recebidos.

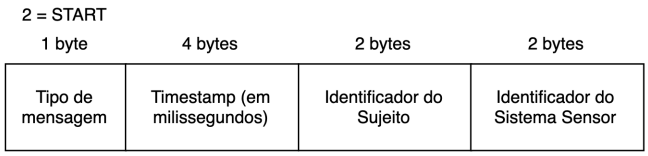

Figura 6: Definição da trama START.

- **STOP** - mensagem enviada de um concentrador para um gestor de serviço a indicar que a recolha de amostras dum sistema sensor particular foi interrompida controladamente (ou seja, foi provocada intencionalmente e não por qualquer condição de erro) [1].

  Para esta trama, definimos 1 *byte* para identificar o tipo de mensagem que esta a ser recebida ou enviada, neste caso para o tipo START, definimos o valor de 3. Os restantes campos da trama foram definidos de acordo com o tipo e tamanho dos dados que estão a ser enviados ou recebidos.

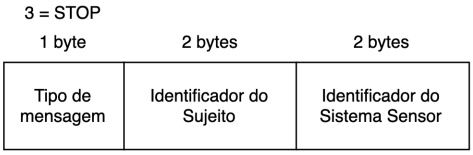

Figura 7: Definição da trama STOP.

4.7. Interpretação das amostras

Com a recolha das amostras dos sensores, procedemos à interpretação dos dados recebidos dos mesmos. Para isso definimos algumas regras que nos ajudam a perceber melhor as condições em que os sujeitos se encontram.

4.7.1. Cálculo da posição do sujeito em repouso

Conhecendo o funcionamento do sensor MPU-6050, e de acordo com a nossa calibração do mesmo, sabemos que se o sujeito estiver em repouso (mas voltado para cima), os valores que devemos receber do acelerómetro devem ser: x=0; y=0; z=1 uma vez que apenas existe uma aceleração, que é no eixo z, a aceleração da gravidade, mas com sentido inverso.

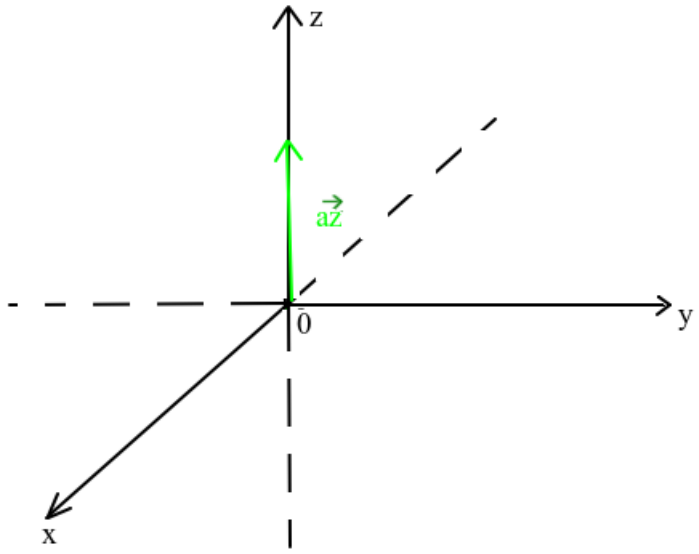

Figura 8: Sujeito em repouso voltado para cima

Se o sujeito estiver em repouso, mas voltado para baixo, a única aceleração a que estará sujeito será a da gravidade, e por isso os valores devem ser x=0;y=0;z=-1

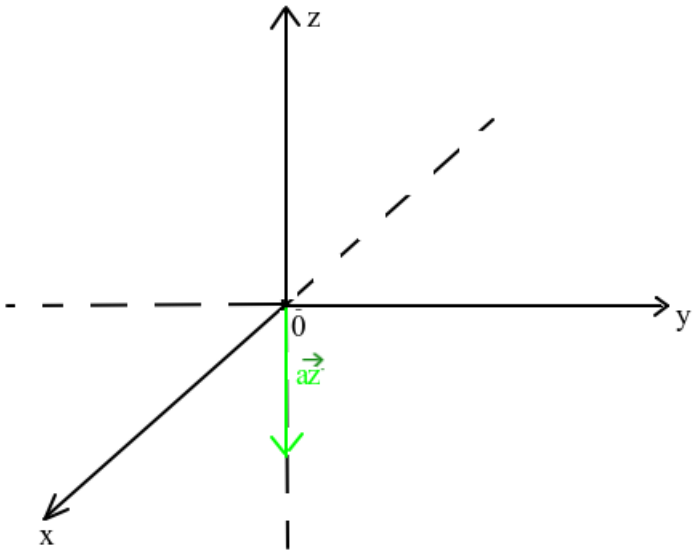

Figura 9: Sujeito em repouso voltado para baixo

Em termos concretos do projeto, a situação real vivida pelo paciente, neste caso (em repouso), está ilustrada na figura 10.

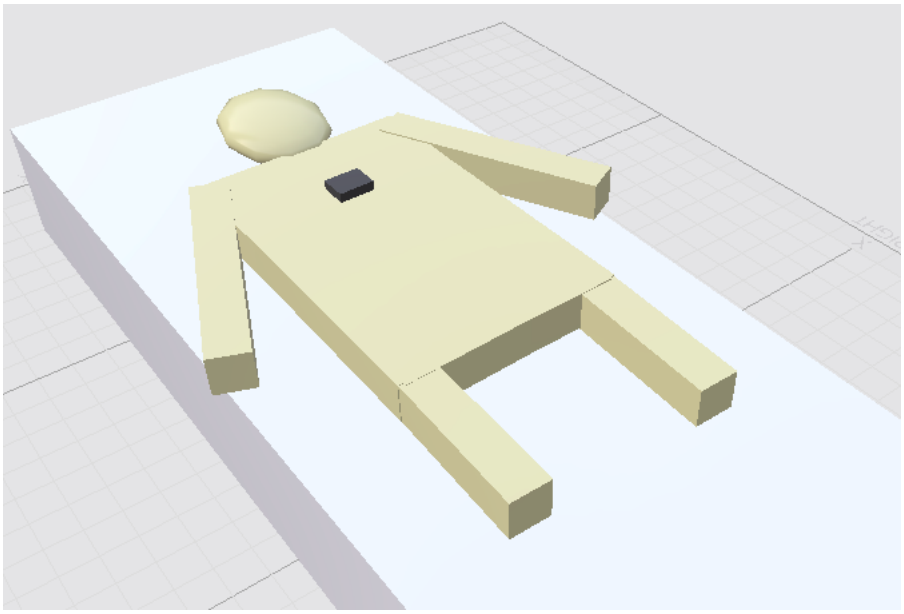

Figura 10: Sujeito em repouso voltado para cima-situação ilustrada

Quando o sujeito se encontra em repouso, poderá-se calcular o ângulo que este faz em relação ao eixo do x (que poderia ser considerado como referência a base onde o sujeito se encontra apoiado).

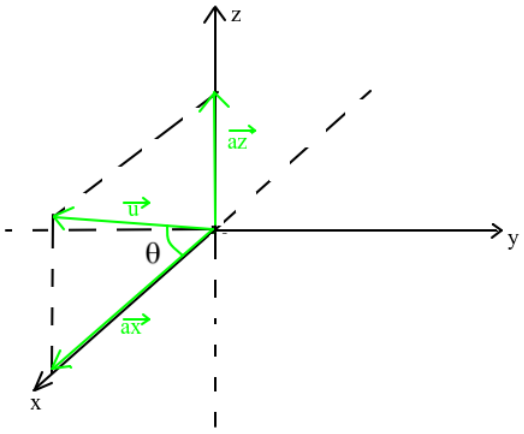

Figura 11: Sujeito em movimento

Sendo que para se calcular esse ângulo teremos que fazer os seguintes cálculos:

Sendo:

$ax = |\vec{u}|.cos\theta$

$az = |\vec{u}|.sin\theta$

$ay = 0$

Temos:

$$\frac{sin\theta}{cos\theta} = tan\theta = \frac{az}{ax} <=>$$

$$<=> \theta = \tan^{-1}\left(\frac{az}{ax} \right)$$

E conseguimos obter o ângulo em que o sujeito encontra, em relação à base de apoio.

Em termos concretos do projeto, a situação real vivida pelo paciente neste caso, está ilustrada na figura 16.

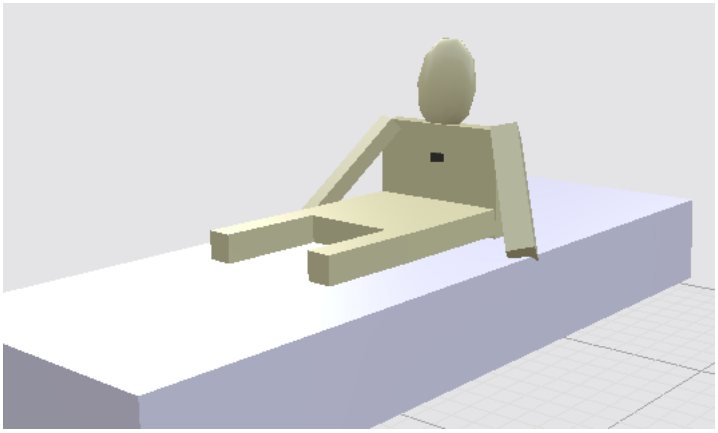

Figura 12: Sujeito em repouso-situação ilustrada

4.7.2. Sujeito em movimento

No caso do sujeito se encontrar em movimento, o módulo do somatório das componentes x, y e z terá um valor diferente de 1g (força gravítica). No cenário deste projeto e no âmbito do sujeito em movimento são referidos os seguintes comportamentos básicos:

- Andamento;
- Agitado;
- Queda.

Como tal será necessária um tratamento das amostras recebidas ao nível da analise do sinal como podemos ver na figura 13.

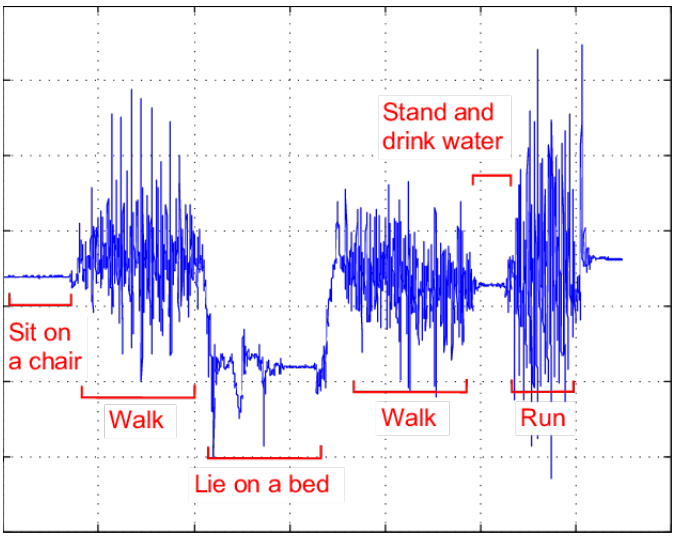

Figura 13: Exemplo de sinal [2].

4.7.2.1 Andamento e Agitado

Quando o sujeito está em andamento ou agitado, a variação do módulo do somatório será maior, pois os valores do acelerómetro que recebe estão sujeitos a maior variação devido há constante mundança de aceleração em relação a um eixo x, y e z quando um sujeito está nessas condições. No entanto, a variação será sempre maior quando está agitado do que quando está em andamento.

Se forem construídos dois gráficos que relacionam os valores obtidos do módulo do somatório com estas duas situações podemos obter semelhantes aos abaixo representados:

Figura 14: Exemplo - Gráfico Andamento [3].

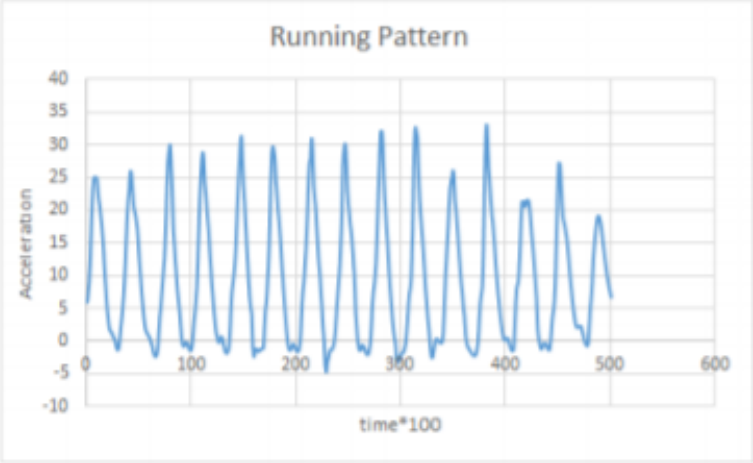

Figura 15: Exemplo - Gráfico Agitado [3].

4.7.2.2 Queda

Quando o sujeito está em queda, o vetor de aceleração vai tender para 0 como se pode ver no exemplo apresentado na figura .

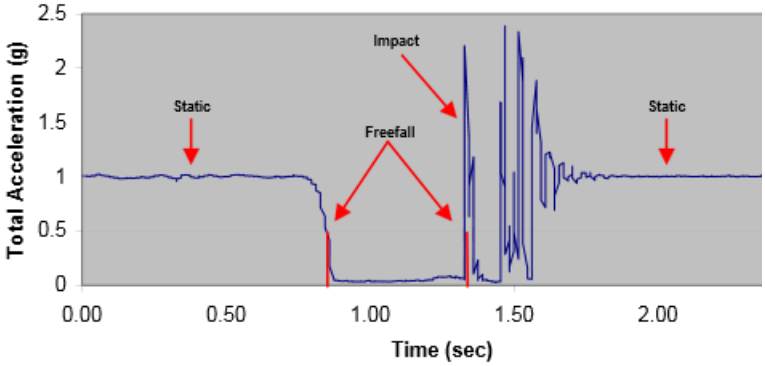

Fig. 1) Drop Test Signature

Figura 16: Exemplo - Gráfico Queda [4].

5. Requisitos

5.1. Requisitos funcionais

Para o correto funcionamento do sistema num todo, é necessário que os seguintes requisitos sejam cumpridos:

- Captação correta dos dados dos sensores, reais ou simulados, pelo concentrador;
- Correta interpretação do comportamento físico dos sujeitos através das amostras recebidas;
- Base de dados bem estruturada e que armazene corretamente todos os dados obtidos pelo gestor de serviço;
- Uma interface de acordo com o objetivo deste sistema e que satisfaça os requisitos do cliente.

5.2. Requisitos não funcionais

Estes requisitos da fase B que serão abaixo enumerados são não funcionais o que indica que não estão diretamente ligados com as funcionalidades do sistema e estão mais relacionados como o tempo de resposta e a fiabilidade. Os requisitos não funcionais definidos são:

- Tempo de cálculo para interpretação da condição do sujeito baixo;
- Fiabilidade da comunicação entre os dispositivos concentradores e os dispositivos sensores;
- Fiabilidade da comunicação entre o gestor de serviço e os dispositivos concentradores;
- Fiabilidade dos dos recolhidos pelos sistemas sensores atuadores;
- Configuração dos parâmetros de obtenção das mensagens pelo gestor de serviço e enviados pelo concentrador.

6. Implementação

Para esta fase B foi preciso implementar novas funcionalidades tais como um gestor de serviço, que comunica com o concentrador já implementado numa fase inicial do projeto e que contém uma base de dados que nos permite guardar todos os dados relativos a comportamentos de sujeitos que possuem e utilizem um sensor.

Este gestor de serviço também foi escrito em linguagem C e a base de dados foi criada usando linguagem MySQL.

6.1. Base de Dados

6.1.1. Código

A base de dados criada consiste em 4 tabelas:

- Concentrador
- Sujeito
- Amostra
- Sistema Sensor

Estas tabelas guardam todos os dados necessários para a interpretação dos dados recebidos do concentrador e de modo a que cada utilizador do serviço consiga visualizar posteriormente cada comportamento de um sujeito através das amostras guardadas para cada um.

A implementação necessária para esta base de dados foi a seguinte:

```
drop database if exists gestorServico;
create database gestorServico;
use gestorServico;
create table concentrador(
id_concentrador integer(10) not null auto_increment,
nome varchar(100),
primary key (id_concentrador)
);
create table sujeito(
id_sujeito integer(10) not null auto_increment,
nome_sujeito varchar(50),
primary key (id_sujeito)
);
create table amostra(
id_amostra integer(10) not null auto_increment,
id_concentrador integer(10),
id_sujeito integer(10),
temperatura float(5),
sensor_simulado binary(2),
acx float(10),
acy float(10),
```

```
acz float(10),
grx float(10),
gry float(10),
grz float(10),
primary key (id_amostra),
foreign key (id_concentrador) references concentrador(id_concentrador),
foreign key (id_sujeito) references sujeito(id_sujeito)
);
create table sistemaSensor(
id_sensor integer(10) not null auto_increment,
sensor_simulado binary(2),
id_sujeito integer(10),
id_concentrador integer(10),
primary key (id_sensor),
foreign key (id_sujeito) references sujeito(id_sujeito),
foreign key (id_concentrador) references concentrador(id_concentrador)
);
delimiter //
create procedure inserir_amostra (in idconcen int , idsujeito int, temp float,ssimulado
    binary, Acx float, Acy float, Acz float, Grx float, Gry float, Grz float)
begin
insert into amostra(id_concentrador, id_sujeito, temperatura, sensor_simulado, acx, acy, acz,
    grx, gry, grz)
        values(idconcen,idsujeito,temp,ssimulado,Acx,Acy,Acz,Grx,Gry,Grz);
end//
delimiter ;
delimiter //
create procedure inserir_sujeito (in nomesujeito varchar(50))
begin
insert into sujeito(nome_sujeito)
        values(nomesujeito);
end//
delimiter ;
delimiter //
create procedure inserir_concen (in nom varchar(100))
begin
insert into concentrador(nome)
        values(nom);
end//
delimiter ;
delimiter //
create trigger inserir_sisSensor
after insert on amostra
for each row
begin
declare idc integer(10);
declare ids integer(10);
declare ssim binary(2);
select id_concentrador
into idc
from amostra
where id_amostra=NEW.id_amostra;
select id_sujeito
into ids
from amostra
where id_amostra=NEW.id_amostra;
select sensor_simulado
into ssim
from amostra
where id_amostra=NEW.id_amostra;
insert into sistemaSensor(sensor_simulado,id_sujeito,id_concentrador)
values(ssim, ids, idc);
end//
delimiter ;
delimiter //
create procedure listar_sensores_sujeito (in idsujeito int)
begin
select id_sensor,sensor_simulado,id_sujeito, id_concentrador from sistemaSensor where
    id_sujeito = idsujeito;
end//
delimiter ;
ALTER TABLE amostra AUTO_INCREMENT=0;
ALTER TABLE sujeito AUTO_INCREMENT=0;
ALTER TABLE concentrador AUTO_INCREMENT=0;
ALTER TABLE sistemaSensor AUTO_INCREMENT=0;
call inserir_sujeito("sujeito1");
call inserir_concen("concentrador1");
call inserir_sujeito("sujeito2");
call inserir_concen("concentrador2");
call inserir_sujeito("sujeito3");
call inserir_concen("concentrador3");
```

Como é possível visualizar, além das tabelas criadas foram também implementados procedimentos e triggers que permitem o bom e lógico funcionamento da base de dados e a interacção externa com a mesma.

6.2. Concentrador

6.2.1. Código

6.2.1.1 Fluxograma

Nesta fase, o código do concentrador foi alterado de modo a que se consiga comunicar com gestor de serviço criado, logo, o fluxograma foi atualizado, sendo representado pela figura 17:

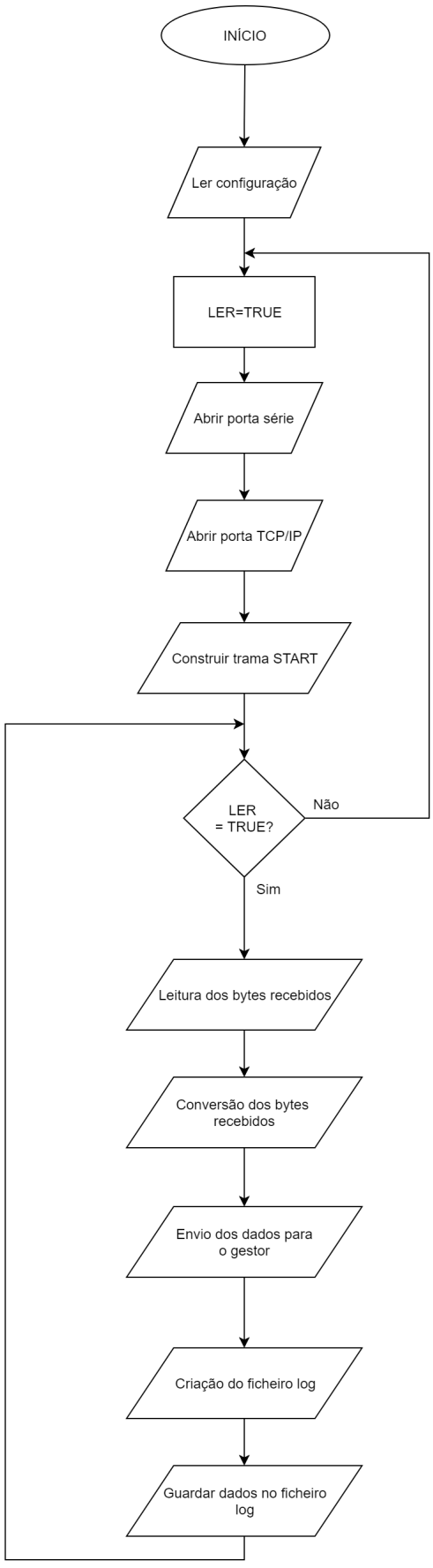

Figura 17: Fluxograma do código Concentrador.

6.2.1.2 Descrição

Para implementação do código no concentrador nesta fase foi preciso introduzir novas bibliotecas que nos permitem realizar a comunicação com o gestor, sendo elas as seguintes:

- "sys/types.h"
- "sys/socket.h"
- "arpa/inet.h"
- "netinet/in.h"
- "sys/time.h"
- "sys/stat.h"

Estas bibliotecas utilizadas permitem-nos abrir *sockets* para comunicação com o gestor de serviço através do protocolo TCP/IP.

Na função *main()* do concentrador são realizadas todas as operações precisas para enviar e receber dados com o sistema sensor ou com o gestor. Numa fase inicial, começa-se por ler o ficheiro de configuração que servirá para criar a trama START, abre-se a porta série para o começo da comunicação, são criados os ficheiros que irão guardar o output do concentrador (log da trama, dados das amostras recebidas e erros recebidos) e é enviada, finalmente, a trama START para o Arduino.

De seguida, quando o concentrador também inicializa a comunicação via TCP/IP para posterior envio de dados ao gestor e entra em ciclo de leitura de dados onde espera pela receção dos bytes relativos aos dados das amostras pedidas.

Depois de recebidas todas as amostras pedidas, elas serão imediatamente enviadas para o gestor para posterior tratamento dos dados e também são guardadas no ficheiro *log*.

No excerto abaixo apresenta-se a função implementada e atualizada do *main()*:

```
int main(int argc, char const *argv[])
{
   int firstRun = 1;
   //armadilhar sinal
   signal(SIGINT, signalhandler);
   readConfig();
   printConfig();
   init();
   int coms = comEnumerate();
   //abrir porta
   openSerial();
   char str[1024];
   char buf[1024];
   int sizeRead;
   //tempo de espera para fazer a calibraao
   sleep(2);
   int sizeStartS = build_packet_start_sensor(str, actualConfig.pm, actualConfig.pa,
       actualConfig.ns);
```

```
//enviar trama de start
//criar log do start
time_t ltime;         /* calendar time */
ltime = time(NULL); /* get current cal time */
sprintf(buf, "Timestamp: %s", asctime(localtime(&ltime)));
write(fdlog, buf, strlen(buf));
comWrite(actualConfig.serialIndex, str, sizeStartS);
//sleep(2);
//ciclo de leitura de valores (amostras ou erro), e criaao de logs
while (1)
{
   // ler da Serial com timeout 5s
   sizeRead = comReadBytes(actualConfig.serialIndex, buf, sizeof(buf), SIZEDATAPACKETEMPTY
       + (actualConfig.ns * SIZEPACKETSAMPLES), 5000);
   if (sizeRead > 0)
   {
      printf("\n"); //printf("SizeRead: %d\n",sizeRead);
      //verificar tipo de trama
      if (buf[0] == DATA)
      {
         if (firstRun == 1)
         {
            actualConfig.ISS = join16(buf + 1);
            actualConfig.ISu = getISu(actualConfig.ISS);
            //send start to manager
            int sizeStartM = build_packet_start_manager(str);
            // open socket
            sockManager = connectSocketManager();
            // send start to manager socket
            send(sockManager, str, sizeStartM, 0);
            firstRun = 0;
            //calculate offsets
            for (int i = 0; i <= 3; i++)
         {
            float val = joinFloat(buf + 16 + (i * 4));
            switch (i % 7)
            {
            case 1: //offset accX
               offX=val-0;
               printf("X: %f %f ",val,offX);
               break;
            case 2: //offset accY
               offY=val-0;
               printf("Y: %f %f ",val,offY);
               break;
            case 3: //offset accZ
               offZ=val-1;
               printf("Z: %f %f\n",val,offZ);
               break;
            }
         }
         }
         printf("DATA ISS: %u ISu: %u Timestamp: %u PA: %u NS: %u\n", actualConfig.ISS,
             actualConfig.ISu, join32(buf + 3), join32(buf + 11), actualConfig.ns);
         sprintf(str, "%u;%u;%u;%u;%u", actualConfig.ISS, actualConfig.ISu, join32(buf + 3),
             join32(buf + 11), actualConfig.ns);
         write(fdout, str, strlen(str));
         //change buf ISS to ISu
         // 0 - TIPO, 1a2 - ISu, 3a6 - TIMESTAMP, 6a9 - PM, 10a13 - PA, 14 - NS, 15(+n*28) a
             42(+n*28) - N1
         split16(buf+1,actualConfig.ISu);
         uint8_t nsT = actualConfig.ns * 7;
         for (int i = 0; i < nsT; i++)
         {
            float val = joinFloat(buf + 16 + (i * 4));
            switch (i % 7)
            {
            case 0: //temp
               printf(" Temp: %f", val);
               sprintf(str, ";%f", val);
               write(fdout, str, strlen(str));
               break;
            case 1: //offset accX
               printf(" RawAccX: %f AccX: %f", val, val - offX);
               val = val - offX;
               splitFloat(buf + 16 + (i * 4),val);
               sprintf(str, ";%f", val);
               write(fdout, str, strlen(str));
               break;
            case 2: //offset accY
```

```
                printf(" RawAccY: %f AccY: %f", val, val - offY);
                val = val - offY;
                splitFloat(buf + 16 + (i * 4),val);
                sprintf(str, ";%f", val);
                write(fdout, str, strlen(str));
                break;
             case 3: //offset accZ
                printf("\nRawAccZ: %f AccZ: %f", val, val - offZ);
                val = val - offZ;
                splitFloat(buf + 16 + (i * 4),val);
                sprintf(str, ";%f", val);
                write(fdout, str, strlen(str));
                break;
             case 4: //gyroX
                printf(" RawGyroX: %f", val);
                sprintf(str, ";%f", val);
                write(fdout, str, strlen(str));
                break;
             case 5: //gyroY
                printf(" RawGyroY: %f", val);
                sprintf(str, ";%f", val);
                write(fdout, str, strlen(str));
                break;
             case 6: //gyroZ
                printf(" RawGyroZ: %f", val);
                sprintf(str, ";%f", val);
                write(fdout, str, strlen(str));
                break;
             default: //restantes
                printf(";%f", val);
                sprintf(str, ";%f", val);
                write(fdout, str, strlen(str));
                break;
             }
          }
          printf("\n");
          write(fdout, "\n", 1);
          // send data to manager
          send(sockManager,buf,16+(28*actualConfig.ns),0);
       }
       else if (buf[0] == ERROR)
       {
          //envia erro para log e sai do ciclo
          printf("ERROR\n");
          int tipErr = buf[7];
          time_t ltime;         /* calendar time */
          ltime = time(NULL); /* get current cal time */
          sprintf(str, "ERROR TYPE: %d Timestamp: %s", tipErr, asctime(localtime(&ltime)));
          write(fderr, str, strlen(str));
          //enviar erro para gestor 0-TP 1a2-ISu 3a4-ISS 5a8-TS 9-TipoErro
          buf[0]=ERROR;
          split16(buf+1,actualConfig.ISu);
          split16(buf+3,actualConfig.ISS);
          split32(buf+5,current_timestamp());
          buf[9]=tipErr;
          send(sockManager,buf,10,0);
          closeFd();
          break;
       }
    }
  }
}
```

Pode-se reparar que foram acrescentadas ao código do concentrador as seguintes novas funções:

- int build_packet_start_sensor(char *str, uint32_t pm, uint32_t pa, uint32_t ns);
- int build_packet_start_manager(char *str);
- int connectSocketManager();

A primeira função referida permite-nos construir uma trama START para envio ao sensor que indica

ao mesmo que pode começar a enviar dados para o concentrador:

```
int build_packet_start_sensor(char *str, uint32_t pm, uint32_t pa, uint32_t ns) // 0-TP,
    1a4-TS, 5a8-PM, 9a12-PA, 13-NS
{
   str[0] = (char)2;
   uint32_t ts = current_timestamp();
   split32(str + 1, ts);
   split32(str + 5, pm);
   split32(str + 9, pa);
   str[13] = ns;
   return 14;
}
```

A segunda função tem a mesma funcionalidade que a primeira mas será para enviar uma trama START ao gestor de serviço de modo a que inicie o seu modo de leitura de dados vindos do concentrador:

```
int build_packet_start_manager(char *str) // 0-TP, 1a4-TS, 5a6-ISu, 7a8-ISS
{
   str[0] = (char)2;
   uint32_t ts = current_timestamp();
   split32(str + 1, ts);
   split16(str + 5, actualConfig.ISu);
   split16(str + 7, actualConfig.ISS);
   return 9;
}
```

Na ultima função referida é realizada a conexão TCP/IP entre o concentrador e o gestor através de sockets específicas que permitem a comunicação com o protocolo pretendido:

```
int connectSocketManager()
{
   struct sockaddr_in address;
   int sock = 0, valread;
   struct sockaddr_in serv_addr;
   if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
   {
      printf("\n Socket creation error \n");
      return -1;
   }
   memset(&serv_addr, '0', sizeof(serv_addr));
   serv_addr.sin_family = AF_INET;
   serv_addr.sin_port = htons(PORT);
   // Convert IPv4 and IPv6 addresses from text to binary form
   if (inet_pton(AF_INET, ADDR, &serv_addr.sin_addr) <= 0)
   {
      printf("\nInvalid address/ Address not supported \n");
      return -1;
   }
   if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
   {
      printf("\nConnection Failed \n");
      return -1;
   }
   return sock;
}
```

6.3. Sistema Gestor de Serviço

6.3.1. Código

6.3.1.1 Fluxograma

Esta nova funcionalidade implementada no projeto pode ser representada pelo fluxograma presente na figura 18.

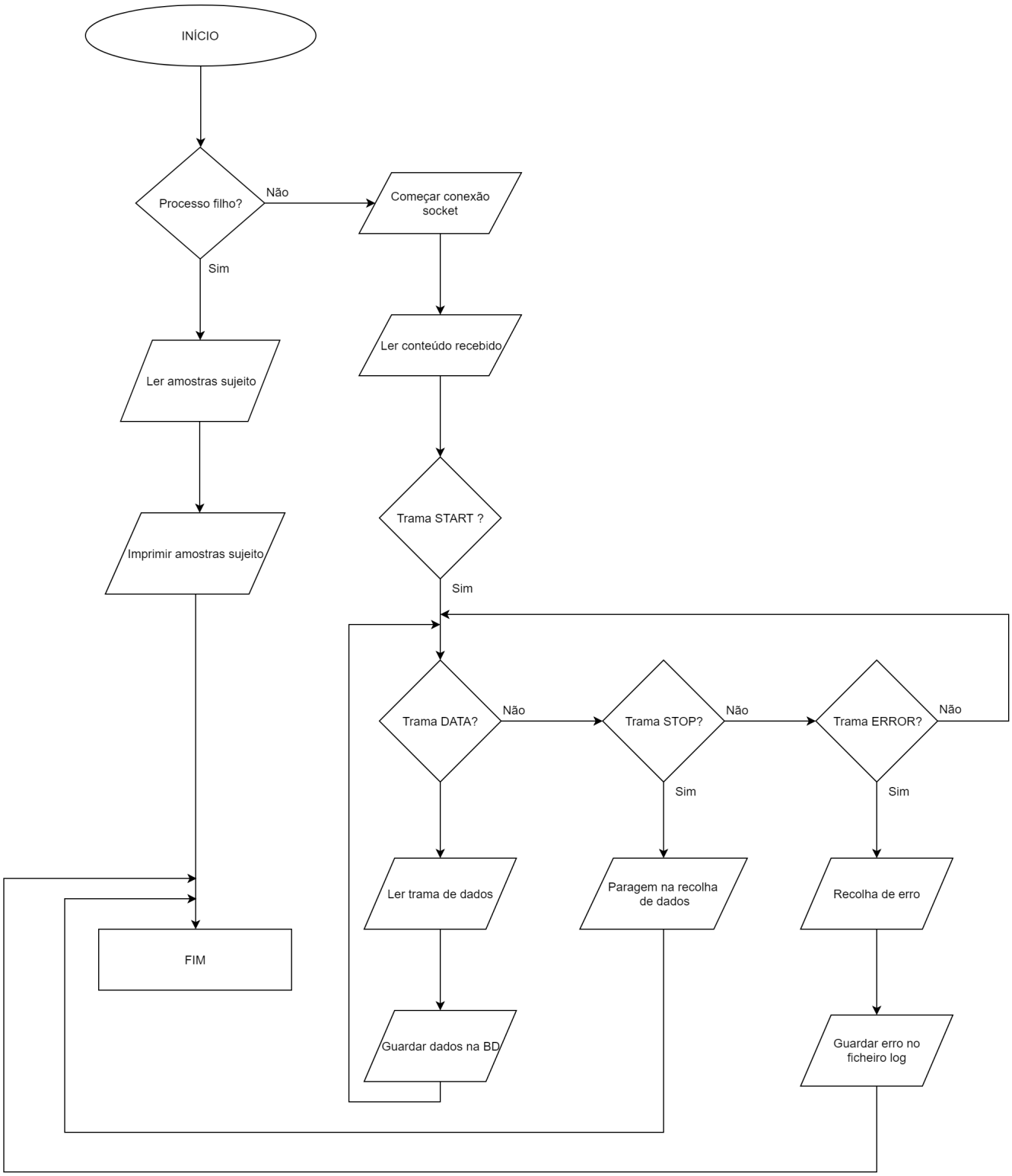

Figura 18: Fluxograma do código Gestor de Serviço.

6.3.1.2 Descrição

Para esta implementação foram adicionadas para além das bibliotecas normais para o funcionamento do gestor e comunicação com o concentrador as seguintes blibliotecas:

- "mysql.h"
- "my_global.h"

Estas bibliotecas permitem-nos conectar o gestor de serviço com a base de dados implementada pois incluí vários métodos já definidos para o efeito.

A função *main()* do gestor realiza todas as operações necessárias para o funcionamento do mesmo e este atua como um servidor sempre à espera por uma ligação vinda do concentrador através de *sockets* TCP/IP para começar a comunicação e recepção de dados.

Quando começa o ciclo de leitura, o gestor armazena todos os dados recebidos na base de dados criada e este ciclo só será terminado quando o gestor receber do concentrador a informação de que o mesmo parou de ler dados de um sensor.

O código da implementação realizada é o seguinte:

```
int main()
{
   readConfig();
   //pipes ui-server
   int pip[2];
   pipe(pip);
   printf("aqui\n");
   // criar fork para UI
   if (fork() == 0)
   { // child UI
      printf("UI process\n");
      close(pip[1]);
      int flags = fcntl(pip[0], F_GETFL, 0);
      if(fcntl(pip[0], F_SETFL, flags | O_NONBLOCK))
         printf("FAIL fcntl\n");// some kind of fail
      float amostras[NAMOSTRAS][7];
      while (1)
      {
         usleep(100);
         // verificar se recebeu do pipe
         int count;
         char buffer[1024];
         count = read(pip[0], buffer, 1024);
         if(count < 0 && errno == EAGAIN) {
            // If this condition passes, there is no data to be read
            for(int i=0; i<actualConfig.numSujeitos; i++){
               printf("Sujeito %d %s\t",actualConfig.sujeito[i], actualConfig.area[i]);
               if(actualConfig.sujeitoAtivos[actualConfig.sujeito[i]]==true){
                  // sujeito ativo
                  amostrasSujeito(actualConfig.sujeito[i], amostras);
                  switch (interpretarAmostra(amostras))
                  {
                  case QUEDA:
                     printf("!QUEDA!\t");
                     break;
                  case AGITADO:
                     printf("!AGITADO!\t");
                     break;
                  case ANDAMENTO:
                     printf("ANDAMENTO\t");
                     break;
```

```
                case PARADO:
                   printf("PARADO\t\t");
                   break;
                }
             }else{
                // sujeito inativo
                printf("!DESLIGADO!\t");
             }
          }
          printf("\n");
       }
       else if(count >= 0) {
          // Otherwise, you're good to go and buffer should contain "count" bytes.
          int suj = atoi(buffer+1);
          if(buffer[0]=='i'){
             actualConfig.sujeitoAtivos[suj]=true;
          }else if(buffer[0]=='o'){
             actualConfig.sujeitoAtivos[suj]=false;
          }
       }
       else {
          // Some other error occurred during read.
          printf("ERRO A LER DO PIPE!\n");
       }
    }
    /* printf("\n AMOSTRAS: \n");
    for(int i=0;i<NAMOSTRAS;i++){
       for(int j=0; j<7; j++){
          printf("%f ",amostras[i][j]);
       }
       printf("\n");
    } */
}
// parent process
else
{
    int server_sockfd, client_sockfd;
    int server_len, client_len;
    struct sockaddr_in server_address;
    struct sockaddr_in client_address;
    server_sockfd = socket(AF_INET, SOCK_STREAM, 0);
    server_address.sin_family = AF_INET;
    server_address.sin_addr.s_addr = htonl(INADDR_ANY);
    server_address.sin_port = htons(PORT);
    server_len = sizeof(server_address);
    bind(server_sockfd, (struct sockaddr *)&server_address, server_len);
    /* Create a connection queue, ignore child exit details and wait for
clients. */
    listen(server_sockfd, 5);
    signal(SIGCHLD, SIG_IGN);
    close(pip[0]);
    while (1)
    {
       char buf[1024];
       // printf("server waiting\n");
       /* Accept connection. */
       client_len = sizeof(client_address);
       client_sockfd = accept(server_sockfd, (struct sockaddr *)&client_address, (socklen_t
           *)&client_len);
       /* Fork to create a process for this client and perform a test to see
whether we're the parent or the child. */
       if (fork() == 0)
       {
          /* If we're the child, we can now read/write to the client on
client_sockfd.
The five second delay is just for this demonstration. */
          //ler conteudo recebido
          int ISu;
          int size = read(client_sockfd, &buf, 1024);
          if (size > 0)
          {
             printf("%d\n", (uint8_t)buf[0]);
             usleep(10);
             //trama de inicio? 0-TIPO, 1a4-TIMESTAMP, 5a6-ISu, 7a8-ISS
             if (buf[0] == (char)START)
             { //trama START
                printf("START\n");
                ISu=join16(buf+5);
                // definir sujeito como ativo
                char bpip[64];
                sprintf(bpip,"i%d",ISu);
```

```
                write(pip[1],bpip,strlen(bpip));
                //TODO:guardar dados de log (TIMESTAMP, ISu,ISS)
                //ciclo de leitura de dados recebidos
                while (1)
                {
                   int size = read(client_sockfd, &buf, 1024);
                   if (size > 0)
                   {
                      // trama de dados? 0 - TIPO, 1a2 - ISu, 3a6 - TIMESTAMP, 7a10 - PM,
                          11a14 - PA, 15 - NS, 16(+n*28) a 43(+n*28) - N1
                      if (buf[0] == DATA)
                      {
                         int ISu = join16(buf + 1);
                         float vals[7];
                         // imprimir dados recebidos
                         //printf("ISu: %d TS: %d PM: %d PA: %d NS: %d \n", join16(buf + 1),
                             join32(buf + 3), join32(buf + 7), join32(buf + 11), buf[15]);
                         for (int j = 0; j < buf[15]; j++)
                         {
                            int c = 0;
                            for (int i = j * 7; i < (j * 7) + 7; i++)
                            {
                               vals[c] = joinFloat(buf + 16 + (i * 4));
                               //printf("%f ", vals[c]);
                               c++;
                            }
                            // guardar dados em bd
                            inserirAmostra(ISu, vals[0], 0, vals[1], vals[2], vals[3], vals[4],
                                vals[5], vals[6]);
                            //
                                printf("%.2f;%.2f;%.2f;%.2f;%.2f;%.2f",vals[1],vals[2],vals[3],vals[4],vals
                            // printf("\n");
                         }
                      }
                      else if (buf[0] == STOP)
                      {
                         //handle stop
                         // definir sujeito como inativo
                         char bpip[64];
                         sprintf(bpip,"o%d",ISu);
                         write(pip[1],bpip,strlen(bpip));
                      }
                      else if (buf[0] == ERROR)
                      {
                         //handle erro
                         // definir sujeito como inativo
                         char bpip[64];
                         sprintf(bpip,"o%d",ISu);
                         write(pip[1],bpip,strlen(bpip));
                      }
                   }
                   else
                   {
                      // definir sujeito como inativo
                      char bpip[64];
                      sprintf(bpip,"o%d",ISu);
                      write(pip[1],bpip,strlen(bpip));
                      close(pip[1]);
                      close(client_sockfd);
                      exit(0);
                   }
                }
             }
          }
          else
          {
             close(client_sockfd);
             exit(0);
          }
       }
       /* Otherwise, we must be the parent and our work for this client is
   finished. */
       else
       {
          close(client_sockfd);
       }
    }
   }
}
```

Como é possível verificar, existem três funções utlizadas essenciais para o tratamento e armazenamento das amostras recebidas:

- void inserirAmostra(int Isu, float temp, int simulado, float Acx, float Acy, float Acz, float Grx, float Gry, float Grz);
- void amostrasSujeito(int id_sujeito, float amostras[NAMOSTRAS][7]);
- int interpretarAmostra(float amostras[NAMOSTRAS][7]);
- int readConfig();

A primeira função permite-nos inserir cada amostra recebida na base de dados e sua implementação é a seguinte:

```
void inserirAmostra(int Isu, float temp, int simulado, float Acx, float Acy, float Acz, float
    Grx, float Gry, float Grz){
   MYSQL conexao;
   char query[2048];
   MYSQL *conn;
   conn = mysql_init(NULL);
   mysql_real_connect(conn, HOST,USER,PASS,DB, 0, NULL, 0);
   sprintf(query, "call inserir_amostra(%d, %d, %f, %d, %f, %f, %f, %f, %f, %f);",Isu, Isu,
       temp, simulado, Acx, Acy, Acz, Grx, Gry, Grz);
   int res=mysql_query(conn, query);
     if (!res) {
        //printf("Registros inseridos %llu\n", mysql_affected_rows(conn));
     }
    else
        printf("Erro na  insero  %d : %s\n", mysql_errno(conn), mysql_error(conn));
   mysql_close(conn);
}
```

Na função amostrasSujeito() é pedido à base de dados que devolva todos os dados requeridos pelo utilizador relativamente a um conjunto de amostras de um sujeito específicado e sua implementação foi a seguinte:

```
void amostrasSujeito(int id_sujeito, float amostras[NAMOSTRAS][7]){
   MYSQL conexao;
   MYSQL_RES *resp;
   MYSQL_ROW linhas;
   MYSQL_FIELD *campos;
   char query[1024];
   sprintf(query,"select temperatura,acx,acy,acz,grx,gry,grz from amostra where id_sujeito =
       %d order by id_amostra desc limit %d;",id_sujeito,NAMOSTRAS);
   int conta; //Contador comum
   mysql_init(&conexao);
   if (mysql_real_connect(&conexao, HOST, USER, PASS, DB, 0, NULL, 0))
   {
      if (mysql_query(&conexao, query))
         printf("Erro: %s\n", mysql_error(&conexao));
      else
      {
         resp = mysql_store_result(&conexao);
         if (resp)
         {
            int i=0;
            while ((linhas = mysql_fetch_row(resp)) != NULL)
            {
               for (conta = 0; conta < mysql_num_fields(resp); conta++){
                  amostras[i][conta]=atof(linhas[conta]);
```

```
            }
            //printf("\n");
            i++;
         }
      }
      mysql_free_result(resp);
   }
   mysql_close(&conexao);
}
else
{
   printf("Conexao Falhou\n");
   if (mysql_errno(&conexao))
      printf("Erro %d : %s\n", mysql_errno(&conexao), mysql_error(&conexao));
}
}
```

Na terceira função referida são interpretados todos os dados das amostras recebidas, ou seja, são identificados os vários comportamentos de um sujeito através das suas respetivas amostras recebidas e, a partir daí, é indicado na interface do utilizador se está em queda, andamento, parado ou agitado.

Para a definição das gamas de valores admitidas em cada comportamento foram realizadas várias experiências com o sensor, e armazenados os valores numa folha de cálculo. Posteriormente o grupo verificou quais os valores que melhor se adequavam a cada comportamento.

De seguida é apresentado a implementação dessa função:

```
int interpretarAmostra(float amostras[NAMOSTRAS][7])
{
   float vetorAcc[NAMOSTRAS];
   vetorAcc[0] = sqrt((amostras[0][1] * amostras[0][1]) + (amostras[0][2] * amostras[0][2]) +
       (amostras[0][3] * amostras[0][3]));
   float menor = vetorAcc[0], maior = vetorAcc[0];
   float vetorGyro[NAMOSTRAS];
   vetorGyro[0] = sqrt((amostras[0][4] * amostras[0][4]) + (amostras[0][5] * amostras[0][5]) +
       (amostras[0][6] * amostras[0][6]));
   float mediaGyro = vetorGyro[0];
   for (int i = 1; i < NAMOSTRAS; i++)
   {
      vetorAcc[i] = sqrt((amostras[i][1] * amostras[i][1]) + (amostras[i][2] * amostras[i][2])
          + (amostras[i][3] * amostras[i][3]));
      vetorGyro[i] = sqrt((amostras[i][4] * amostras[i][4]) + (amostras[i][5] *
          amostras[i][5]) + (amostras[i][6] * amostras[i][6]));
      if (vetorAcc[i] > maior)
      {
         maior = vetorAcc[i];
      }
      if (vetorAcc[i] < menor)
      {
         menor = vetorAcc[i];
      }
      mediaGyro += vetorGyro[i];
      if (vetorAcc[i - 1] < 0.6)
      {
         if (vetorAcc[i] < vetorAcc[i - 1])
         {
            return QUEDA;
         }
      }
   }
   float amplitudeAcc = maior - menor;
   mediaGyro = mediaGyro / NAMOSTRAS;
   // printf("Acc:%.2f Gyro:%.2f\n", amplitudeAcc, mediaGyro);
   if (amplitudeAcc > 1.6 && mediaGyro > 150)
   {
      // printf("\nAGITADO\n");
      return AGITADO;
   }
   else if (amplitudeAcc > 0.5 && mediaGyro > 50)
   {
      // printf("\nANDAMENTO\n");
      return ANDAMENTO;
```

```
    }
    else
    {
        // printf("\nPARADO\n");
        return PARADO;
    }
}
```

Na função *readConfig()* é implementado a leitura e interpretação do ficheiro de configuração do Sistema Gestor de Serviço, onde constam todos os identificadores do sujeito e respetivas áreas associadas.

Tal foi implementado no seguinte código:

```
int readConfig()
{
    FILE *fp;
    char *line = NULL;
    size_t len = 0;
    ssize_t read;
    fp = fopen(CONFIG_GESTOR, "r");
    if (fp == NULL)
        return -1;
    read = getline(&line, &len, fp);
    printf("%s\n", line);
    int argSize = 0;
    int tc = NONE;
    char *token = strtok(line, ";");
    // Keep looping while one of the
    // delimiters present in str[].
    while (token != NULL)
    {
        if ((argSize % 2) == 0)
        {
            // id sujeito
            actualConfig.sujeito[actualConfig.numSujeitos] = atoi(token);
        }
        else
        {
            // nome area
            strcpy(actualConfig.area[actualConfig.numSujeitos], token);
            printf("%d;%s\n", actualConfig.sujeito[actualConfig.numSujeitos],
                actualConfig.area[actualConfig.numSujeitos]);
            actualConfig.numSujeitos++;
        }
        argSize++;
        token = strtok(NULL, ";");
    }
    fclose(fp);
    if (line)
        free(line);
    return 0;
}
```

7. Testes e análise de resultados

Neste capítulo iremos apresentar os vários testes realizados ao sistema de modo a perceber se todos os dados recebidos pelo gestor são bem interpretados e que estão de acordo com o pretendido para este projeto garantindo a fiabilidade de toda a comunicação.

7.1. Execução da interface de utilizador

Para testar o correto funcionamento da interface de utilizador, procedemos à execução da mesma tendo podido ser observado que aconteceram as situações previstas de "PARADO", "ANDAMENTO", "QUEDA"e "AGITADO", tal como mostra a seguinte figura.

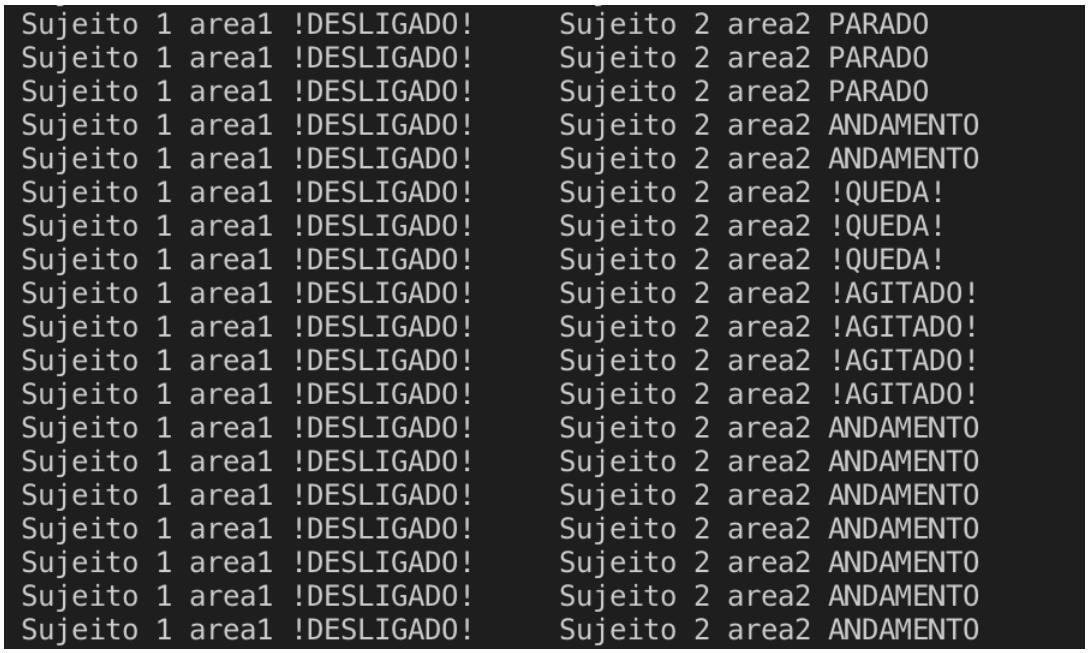

Figura 19: Gráfico exemplo dos dados recebidos.

Além disso, é possível observar também o uso de vários concentradores/sujeitos que no caso exemplificado como apenas estava a ser monitorizado o sujeito 2, apenas deste se obtiveram dados, sendo que para verificar se os dados estavam corretos verificámos estes dados com os do concentrador nos mesmos instantes de tempo.

7.2. Visualização gráfica dos dados recolhidos

Usando o programa GnuPlot pudemos analisar em tempo real os dados recolhidos pelo concentrador, tal como mostra a figura 20, uma vez que o Sistema Sensor escreve para dois ficheiros os valores utilizados na interpretação das amostras.

O *script* de GnuPlot que usámos para elaboração do gráfico foi o seguinte:

```
set grid
set title 'Grafico'
set xlabel 'tempo (em milis)'
set datafile separator ","
set format y "%.2f"
plot 'graphAcc.csv' u 1:2 w lp t 'acelerometro', 'graphGyro.csv' u 1:2 w lp t 'giroscopio'
pause 1
reread
replot
```

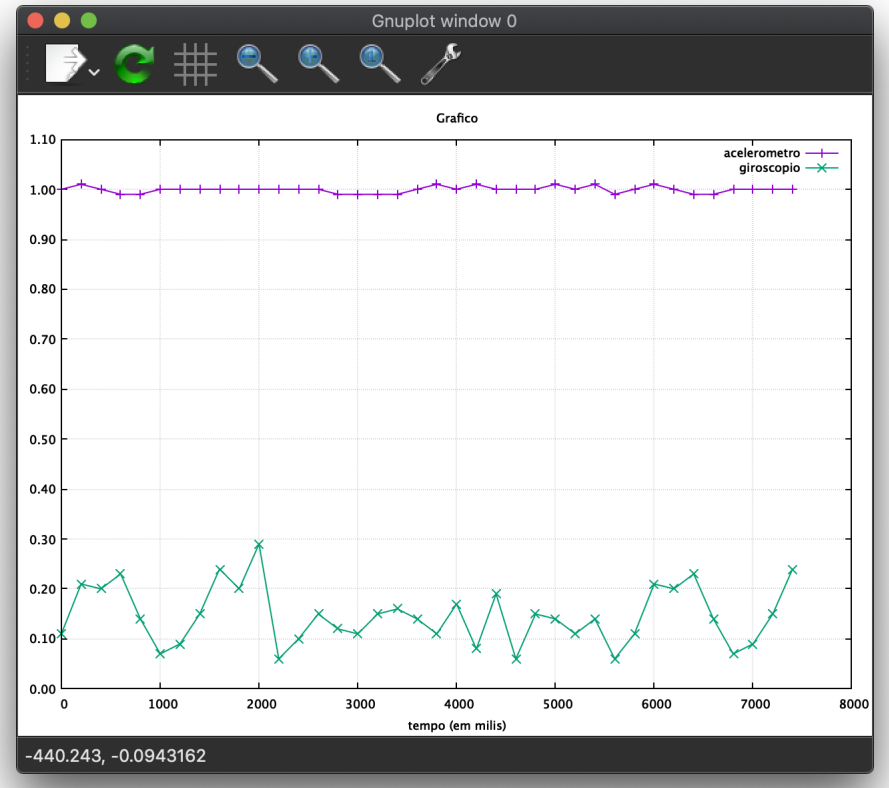

Figura 20: Gráfico exemplo dos dados recebidos.

8. Conclusão

Após a conclusão desta fase B, o grupo fica satisfeito com o desempenho realizado, uma vez que os objetivos traçados foram cumpridos e foi um sucesso a implementação de um gestor de serviço e uma comunicação estável e fiável do mesmo com o concentrador.

Mais uma vez, o cumprimento rigoroso das tarefas atribuídas a cada elemento do grupo e o empenho de cada um foi crucial para ultrapassar dificuldades em algumas noções novas que eram precisas para este projeto, tais como conhecimento de protocolos de ligação ou mesmo o estudo de novas bibliotecas para a linguagem utilizada que nos permitiu trazer novas funcionalidades pedidas para esta fase.

O código do concentrador foi aperfeiçoado e já possui a implementação de módulos independentes de leitura e escrita, um requisito pretendido pelo grupo na fase anterior que não desenvolveu e que se propôs a realizar nesta fase B.

Para concluir, fica por terminar o desenvolvimento do sensor simulado que já está em modo protótipo e é algo que o grupo pretende acabar já na próxima fase.

9. Referências

[1] **Docentes de Laboratórios de Telecomunicações e Informática II**, *"Sistema de Monitorização de Atividade Física - ANEXO 3 – Fase B"*, Universidade do Minho, 2019

[2] **Zhang, Mi A. Sawchuk, Alexander.** *"Proceedings of the 2012 ACM Conference on Ubiquitous Computing"*, 2012

[3] **Bayat, Akram Pomplun, Marc Tran, Duc.** *"A Study on Human Activity Recognition Using Accelerometer Data from Smartphones"*. Procedia Computer Science, 2014.

[4] **"App note: Free-fall sensing for drop-force modeling using a Kionix MEMS tri-axis accelerometer « Dangerous Prototypes»", Dangerousprototypes.com**

Disponível em http://dangerousprototypes.com/blog/2014/09/21/app-note-free-fall-sensing-for-drop-force-modeling-using-a-kionix-mems-tri-axis-accelerometer [Acedido em em 1 de abril de 2019]