

# Sistema de Monitorização de Atividade Física

---

Fase A

**Grupo 3**

18 de fevereiro de 2019



# Índice:

Conteúdo .....	1
Constituição do Grupo .....	4
Introdução .....	5
Ferramentas .....	6
Hardware.....	6
Software .....	10
Calendarização .....	11
Enquadramento do projeto.....	12
Sistema Sensor .....	13
Sensor.....	15
Arduíno.....	16
Concentradores.....	17
Comunicação do Sistema .....	19
Arquitetura.....	20
Arquitetura do software Arduino-Sensor .....	22
Arquitetura do software Concentrador .....	28
Estrutura do protocolo de dados .....	32
Tipos de Mensagens.....	33
1. DATA .....	33
2. ERROR .....	33
3. START .....	34
4. STOP .....	34
5. READY.....	34
Formato dos valores recolhidos.....	35
Testes e análises de resultados.....	36
Conclusão .....	38
Referencias bibliográficas .....	39



# Índice de Figuras

---

Figura 1 - ESP32-WROOM-32D Specifications .....	8
Figura 2 - Pinos ESP32-WROOM-32D .....	8
Figura 3 - Pinos do sensor .....	9
Figura 4 - Diagrama de Gantt .....	11
Figura 5 - Arquitetura do Sistema de Monitorização de Atividade Física .....	12
Figura 6 - Sistema de ligação ESP32 + MPU6050 .....	14
Figura 7 - Sensor MPU6050 .....	15
Figura 8 - Diagrama de Sequência, da comunicação entre sistema sensor e concentrador .....	19
Figura 9 - Sistema sensor de Atividade Física .....	20
Figura 10 - Sistema Sensor .....	21
Figura 11 - Algoritmo do código no Arduino: processamento de recolha/envio de informação .....	22
Figura 12 - Bibliotecas usadas no Arduino .....	23
Figura 13 - Estabelecimento de conexão WIFI, ligação à porta através do protocolo UDP e envio da trama READY .....	23
Figura 14 – Código do Arduino com leitura de trama e envio .....	24
Figura 15 - Função START, com interpretação da trama e procura de erros .....	25
Figura 16 - Função DATA .....	25
Figura 17 - Função ERROR .....	26
Figura 18 – Dados de configuração do access point .....	26
Figura 19 – Conexão à porta através do protocolo UDP .....	27
Figura 20 - Diagrama de blocos do concentrador (em linguagem natural) .....	28
Figura 21 – Bibliotecas usadas no código arduino .....	30
Figura 22 - Método usado para obter valores dos eixos do acelerómetro .....	30
Figura 23 - calibração giroscópio .....	31
Figura 24 - Funcionamento do Protocolo .....	32






# Índice de Tabelas

---

Tabela 1 - Hardware .....	6
Tabela 2 - ESP32-WROOM-32D .....	7
Tabela 3 - Descrição dos Pinos do sensor .....	9
Tabela 4 - Software .....	10
Tabela 5 - Ligações entre ESP32 e modulo MPU6050.....	21
Tabela 6 - Valores obtidos a partir do sensor Acelerómetro .....	37

## Constituição do Grupo

---

	Augusto Mota (a76563) a76563@halunos.uminho.pt
	Hugo Machado (a80362) a80362@alunos.uminho.pt
	Miguel Moreira (a77314) a77314@alunos.uminho.pt



# Introdução

---

No âmbito da UC (Unidade Curricular) de LTI II (Laboratórios de Telecomunicações e Informática II) foi proposto aos elementos de cada grupo a realização de um projeto com o tema “Sistema de Monitorização de Atividade Física”.

Este projeto tem como foco principal a implementação de um sistema que possibilita a monitorização de atividade física de doentes internados em instituições de saúde ou apoio social.

Nesta primeira fase o grupo terá como objetivo o planeamento, desenvolvimento e teste de todo o hardware e software necessários. Em cada área do sistema existirá um conjunto de sensores que irão obter e registar valores, num determinado tempo real, da atividade física de cada paciente. Para tal, cada paciente possuirá um sistema sensor para obter os respetivos valores.

# Ferramentas

Para a realização deste projeto temos ao nosso dispor diversos recursos de modo a ser possível a finalização do projeto prático com sucesso. Alguns destes recursos são classificados como recursos físicos constituindo deste modo a componente hardware a ser utilizada e outros, os sem formato físico que serão os de Software.

## Hardware






Imagem	Designação	Descrição
	ESP32-WROOM-32D	Componente responsável pelo processamento dos dados
	MPU-6050	Acelerómetro e Giroscópio para obtenção de dados relativos à atividade física
	Jumpers	Comunicação série entre Arduinos e MPU-6050
	Computador	Desenvolvimento de código e relatórios
	Breadboard	Interface de conexão entre os circuitos

Tabela 1 - Hardware



## Lista de componentes adquiridos:

- ESP32-WROOM-32D

Módulo genérico de Wi-Fi + BT + BLE MCU poderosos que apontam para uma ampla variedade de aplicações como por exemplo redes de sensores com baixa potencia, codificação de voz, *streaming* de músicas e decodificação de MP3.

Module:	ESP32-WROOM-32D:
SPI flash	32 Mbits, 3.3 V
Core	ESP32-D0WD
Crystal	40 MHz
Antenna	U.FL connector (which needs to be connected to an external IPEX antenna)
Dimensions(Unit: mm)	$(18.00 \pm 0.10) \times (19.20 \pm 0.10) \times (3.20 \pm 0.10)$

Tabela 2 - ESP32-WROOM-32D

O componente tem integrado Bluetooth, Bluetooth LE e Wi-Fi, garantindo uma ampla gama de aplicativos que estejam á sua volta: a utilização do Wi-Fi permite um grande alcance físico e conexão direta com a internet usando um Wi-Fi router, enquanto que utilizando Bluetooth permite apenas conectar convenientemente ao telefone ou *broadcast* de baixa energia. A *sleep current* do chip ESP32 é inferior a 5 micros Amperes, tornando perfeito para aplicações eletrónicas alimentadas por baterias e aparelhos *wearable*. Este chip suporta uma taxa de dados até 150 Mbps e 20 dBm de potencia de saída de antena para ser possível e garantido o maior alcance físico. O modulo garante o melhor desempenho para integração eletrónica, alcance, consumo de energia e conectividade. Este sensor teve um custo de 12.99 euros.



Categories	Items	Specifications
Wi-Fi	Protocols	802.11 b/g/n (802.11n up to 150 Mbps) A-MPDU and A-MSDU aggregation and 0.4 $\mu$ s guard interval support
	Frequency range	2.4 GHz ~ 2.5 GHz
Bluetooth	Protocols	Bluetooth v4.2 BR/EDR and BLE specification
	Radio	NZIF receiver with -97 dBm sensitivity Class-1, class-2 and class-3 transmitter AFH
	Audio	CVSD and SBC
Hardware	Module interfaces	SD card, UART, SPI, SDIO, I <sup>2</sup> C, LED PWM, Motor PWM, I <sup>2</sup> S, IR, pulse counter, GPIO, capacitive touch sensor, ADC, DAC
	On-chip sensor	Hall sensor
	Integrated crystal	40 MHz crystal
	Integrated SPI flash <sup>1</sup>	4 MB
	Operating voltage/Power supply	2.7 V ~ 3.6 V
	Operating current	Average: 80 mA
	Minimum current delivered by power supply	500 mA
	Recommended operating temperature range <sup>2</sup>	-40 °C ~ +85 °C

Figura 1 - ESP32-WROOM-32D Specifications

De seguida encontrasse disponível o pin layout do componente ESP32-WROOM-32D para uma melhor compreensão do componente.

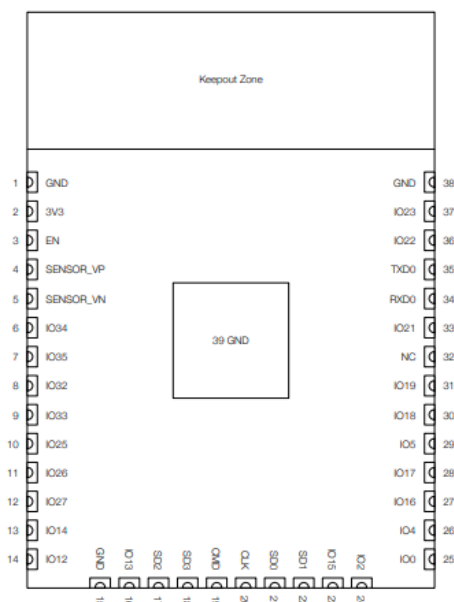


Figura 2 - Pinos ESP32-WROOM-32D

- MPU-6050

O componente MPU-6050 contém um giroscópio de 3 eixos (x,y,z) e também um acelerômetro de 3 eixos (x,y,z) permitindo medições de ambos de uma forma independente, mas ambos baseados nos mesmos eixos, eliminando problemas de erros de eixo ao usar dispositivos separados.

Item:	MPU-6050:
VDD	2.5V±5%, 3.0V±5%, or 3.3V±5%.
VLOGIC	1.71V to VDD
Serial Interfaces Supported	I <sup>2</sup> C
Pin 8	VLOGIC
Pin 9	AD0
Pin 23	SCL
Pin 24	SDA

Tabela 3 - Descrição dos Pinos do sensor

Este sensor fornece nove eixos de deteção do movimento, composto por:

- Acelerômetro de três eixos de saída digital com escala programável de +- 2g, +-4g, +-8g e +-16g
- Compasso de tri-eixo com saída digital com uma faixa de escala completa de +- 1200 micros T.
- Sensores de taxa angular de saída digital X, Y e Z(giroscópios) com uma taxa de escala total programável pelo usuário de +-250, +-500, +-1000 e +-2000° / s.

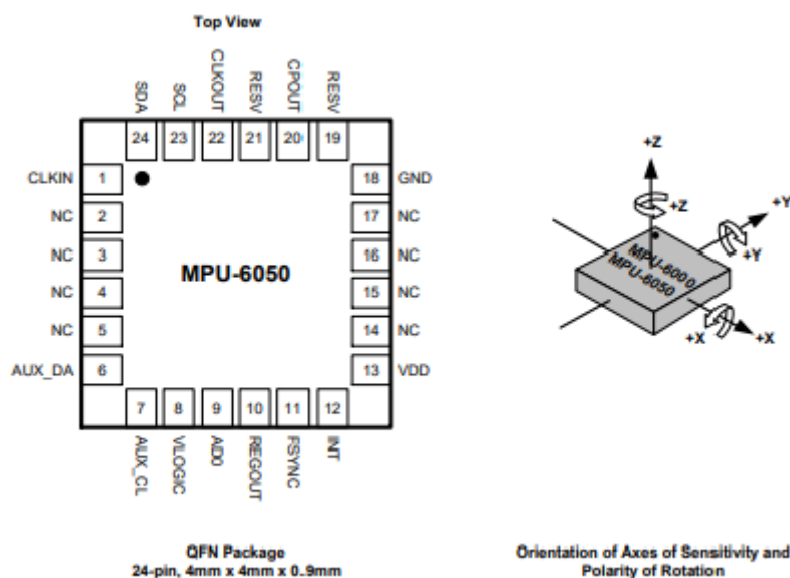


Figura 3 - Pinos do sensor

## Software

Logotipo	Nome	Funcionalidade
	Arduino IDE	Programar o módulo Arduino
	Facebook	Comunicação entre membros do grupo
	Google Chrome	Motor de pesquisa
	Gantt Project	Planificação temporal do projeto
 Word	Word	Edição de texto
	Atom	Editor de texto de código

Tabela 4 - Software



# Calendarização

Para demonstrar como foi feita a calendarização da fase A, a melhor forma que o grupo encontrou de calendarizar as tarefas foi através de Mapas de Gantt.

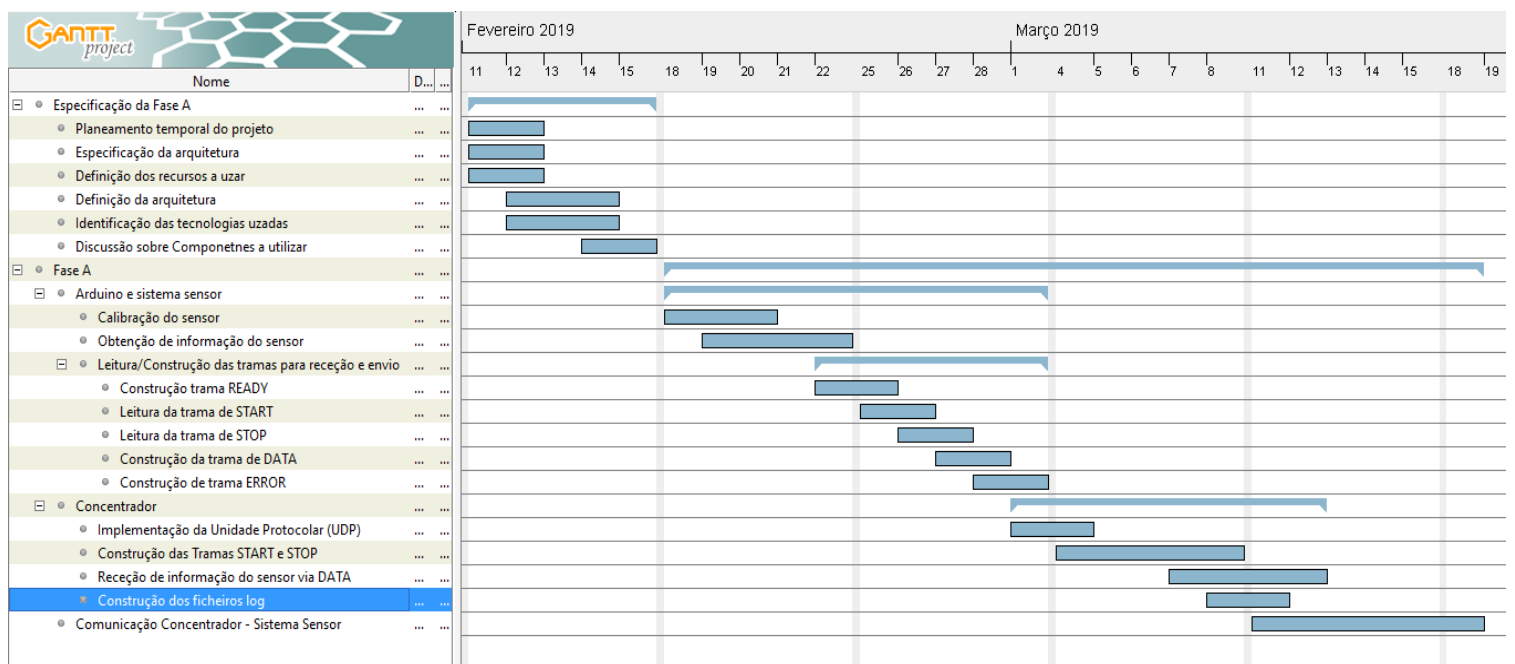


Figura 4 - Diagrama de Gantt

# Enquadramento do projeto

De seguida apresentamos a arquitetura que pretendemos desenvolver ao longo deste projeto:

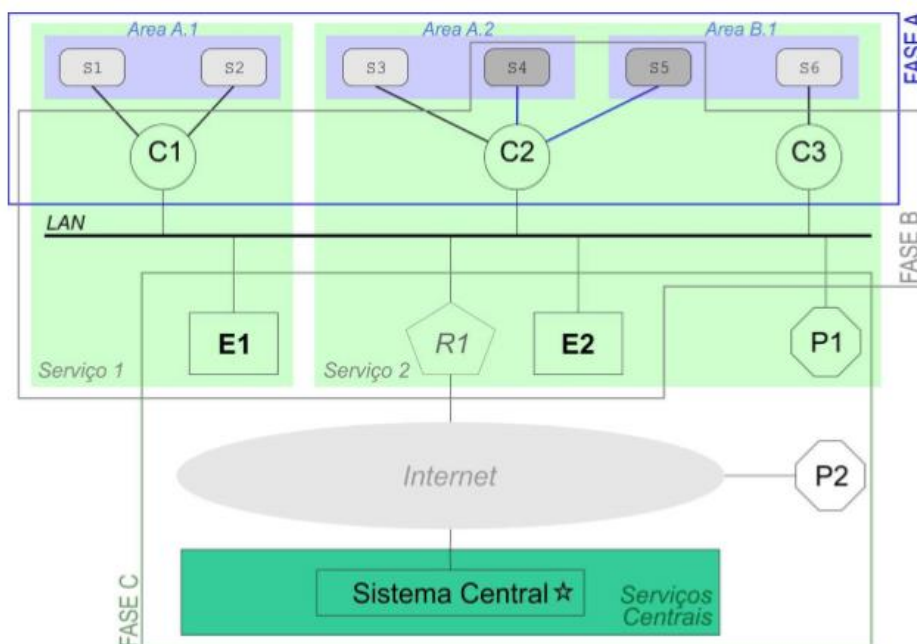


Figura 5 - Arquitetura do Sistema de Monitorização de Atividade Física

Este sistema sensor que vamos realizar, visa a implementação de um sistema que seja possível monitorizar a atividade física dos doentes. Este sistema, só é possível realizar recorrendo a uma implementação de uma rede de sensores que nos vai possibilitar recolher valores necessários. Esta recolha de dados vai ser realizada através do Arduino que de seguida envia essa informação para o concentrador, tendo funções de processamento de dados e deteção de erros.

Todos os dados que vamos recolher ao longo do tempo serão enviados para um certo servidor que tem uma capacidade de armazenar a informação numa base de dados para depois podermos consultar esses valores.

Será implementada também neste projeto uma interface Web, permitindo a todos utilizadores proceder à consulta de informação através de qualquer equipamento que tenha acesso à Internet.



## Sistema Sensor

1. Cada sistema sensor vai ter a capacidade de recolher/medir valores da atividade física e enviar esses valores/amostras (medições) em certos intervalos adequados do sistema para o concentrador, onde de seguida os concentradores vão recolher as amostras dum certo conjunto de sensores.
2. O sistema sensor que vamos realizar é constituído por um dispositivo sensor acelerómetro de três eixos e também por um equipamento de recolha de amostras que serão enviadas para o concentrador em tempo real.
3. Neste sistema vamos utilizar um dispositivo *wearable* constituído por um sensor acelerómetro de 3 eixos (x,y,z) com o objetivo de extrair informação que seja relevante á atividade física do paciente. Dispositivos *wearable*, ou "tecnologia vestível", engloba todos os equipamentos em termos eletrónicos que contem os seus próprios processadores e podem ser utilizados como acessórios ou peças de roupa.
4. Temos também a autorização para integrar outro tipo de sensores como por exemplo giroscópios e magnetómetros se forem uteis para a monitorização da atividade física.
5. Para a implementação hardware do dispositivo sensor foi recomendado a utilização do módulo que contem o circuito integrado, *MPU-6050*, que integra um acelerómetro contendo 3 eixos (x,y,z) e um giroscópio de 3 eixos.
6. Em termos de placa do microcontrolador foi recomendado a aquisição de um módulo Arduino ou um dispositivo que seja compatível com a plataforma Arduino.
7. Cada sistema sensor terá uma *tag* única de identificação ISS (Identificação de Sistema Sensor).
8. Os valores das amostras retirados através dos dispositivos sensor devem ser identificados com o seu próprio ISS, *timestamps* (etiquetados temporalmente) quando são enviados para o concentrador.

Na figura 6, podemos ver a arquitetura do circuito que implementamos para a realização do projeto que nos foi proposto. O circuito como foi dito anteriormente é composto pelo módulo MPU-6050, estando ligado a uma entrada analógica do sensor ESP-32 contendo transmissão Wi-Fi e Bluetooth e contendo um microcontrolador integrado. Basicamente, o módulo vai retirar valores para serem enviados através do módulo ESP-32.

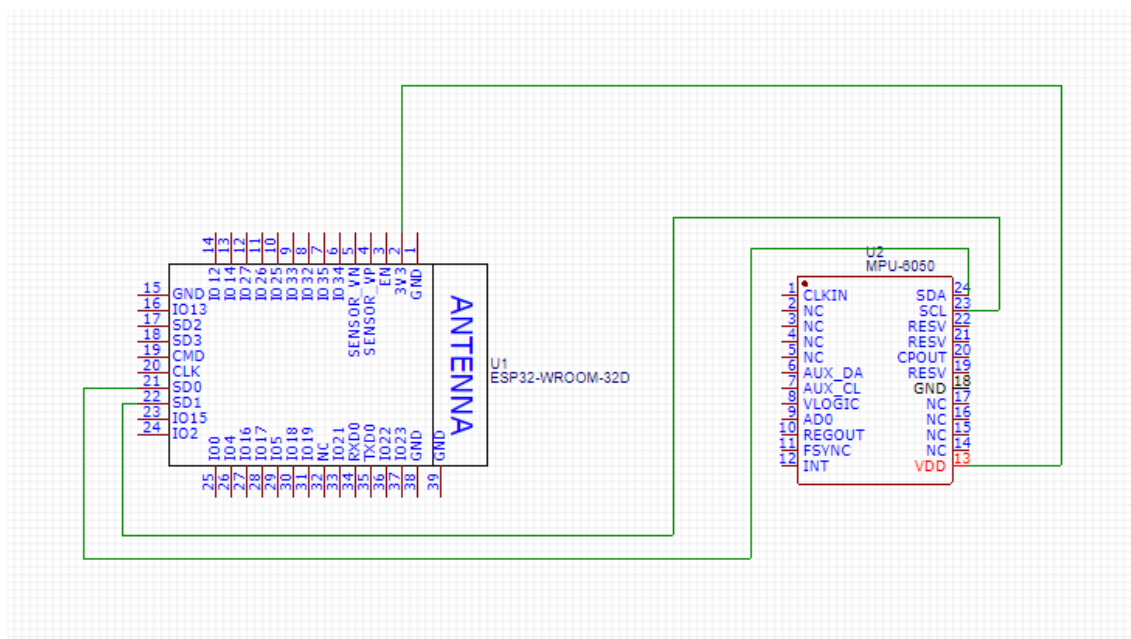


Figura 6 - Sistema de ligação ESP32 + MPU6050

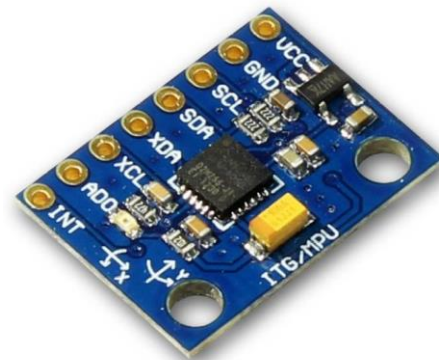


## Sensor

O grupo escolheu trabalhar com o sensor MPU6050, principalmente porque foi o sugerido pelos docentes da cadeira, mas também pela quantidade de suporte *online* e pela facilidade de trabalho.

Este tem também um chip digital de baixo consumo de energia e regulador de fonte de alimentação que apresenta alta sensibilidade e baixa interferência de ruído. Ainda assim pode ser configurado para diferentes níveis de sensibilidade à aceleração e pode usar diferentes faixas de medição de taxa angular. Ele é fornecido com um SDK (Software Development Kit) detalhado, o que torna o processo de prototipagem rápido e fácil.

Contudo este produto é bom para aplicações em que inclinação, movimento e sensorimento de toque são necessários.



### Figura 7 - Sensor MPU6050





## Arduíno

1. Em termos de código, o Arduíno vai ser identificado por um ISS que vai ser definido no próprio código do Arduíno (*hardcoded*), e todos os sistemas sensores também devem ter atribuídos códigos únicos.
2. O Arduíno vai enviar uma mensagem de dados (*DATA*) periodicamente a um ritmo definido por um parâmetro de configuração denominado de período (Milissegundos) entre as mensagens de dados (*PM*) que vão ser recebidos no concentrador. Se este parâmetro *PM* recebido no concentrador for igual a zero, temos que assumir um valor por defeito definido no próprio Arduíno (*hardcoded*).
3. A mensagem de dados tem de conter um conjunto de valores de *NS* amostras que vão ser recolhidas pelos dispositivos sensores num determinado período de Amostragem (*PA*) que vai ser recebido no concentrador. Se estes valores *NS* e *PA*, forem nulos quando são recebidos pelo concentrador, devemos assumir um valor por defeito que vai ser definido no próprio Arduíno (*hardcoded*).
4. O início de envio de mensagens vai ser iniciado quando é recebido um *START* vindo do concentrador e sua interrupção é feita quando recebemos uma mensagem *STOP* que vem também do concentrador.
5. Quando o Arduíno não tem capacidade de enviar mensagens de dados ao ritmo definido pelos parâmetros *NS*, *PM* e *PA* será enviada uma mensagem de *ERROR*. OS valores otimizados para *NS*, *PM* e *PA* deverão ser devidamente estimados durante o teste e desenvolvimento de código do Arduíno.



## Concentradores

1. Em cada área vamos encontrar 1 ou mais concentradores.
2. Os concentradores têm de ter a capacidade de validar e normalizar os valores (processamento de valores).
3. Os concentradores vão ser capazes de realizar uma interpretação básicas das amostras (Associar a um conjunto de valores das amostras do comportamento físico dos pacientes) e de seguida é enviado o resultado da atividade física dos sujeitos para um sistema de Gestor Serviço.
4. Os concentradores devem gravar os valores das amostras em ficheiros tipo *log*.
5. A recolha de dados no concentrador tem que integrar uma fase de calibração dos acelerómetros (valores máximos de cada eixo tem de ser +1 e os seus valores mínimos deve ser -1).
6. A interface de configuração no próprio concentrador deve registar a área onde o paciente vai ser instalado e a associação entre cada sistema sensor que o paciente esta ligado e o respetivo sujeito a monitorizar. Na Fase A só é necessário cada sujeito ser identificado por uma *tag* única de *ISu* (Identificação do Sujeito).
7. Quando recebemos os valores nos concentradores eles devem ser:
  - Validados;
  - Cada valor retirado deve ser associado ao respetivo *ISu*;
  - Armazenamento em ficheiros log.
8. É essencial criar uma interface de visualização de valores das amostras e validados para cada sujeito/sistema dispositivo-sensor tanto em tempo real como num intervalo temporal pré-definido (ou definir pelo utilizador).
9. O código que vamos desenvolver vai ser realizado em linguagem C.
10. No concentrador temos de criar vários módulos que vão ser independentes:
  - Interface de utilização;
  - Arranque de sistema consultando um ficheiro de configuração;
  - Tratamento de comunicações.



11. O concentrador deverá utilizar um ou mais ficheiros log para registar os vários procedimentos que vão ser executados ao longo do tempo:
  - Arranque;
  - Setup das portas de comunicação;
  - Recolha de dados dos dispositivos sensores;
  - Registar os erros que vão surgindo ao longo da comunicação.
12. Um concentrador tem a capacidade de poder ter ligados vários sistemas sensores, e em cada sistema sensor com os seus dispositivos sensores reais (Arduíno + Acelerómetro/Giroscópio) que serão conectados via WIFI.
13. Em termos de modulo de arranque, este deverá ler um ficheiro de configuração com as suas configurações necessárias para a definição dos parâmetros de comunicação com as portas que vão ser utilizadas nas ligações aos sistemas sensores (USB, UDP) e também dos seus parâmetros NS, PM e PA.
14. O módulo de arranque tem a capacidade de guardar em log o momento de arranque e quaisquer resultados anormais da análise dos dados do ficheiro.
15. Tendo um arranque normal, o modulo de comunicações tem de tratar todas as interações entre o concentrador e os vários sistemas de sensores conectados, e quando recebe mensagens de dados que são validados com os valores de amostras sendo esses valores armazenados numa área temporária para depois o modulo de armazenamento e de processamento ter acesso aos dados. Este modulo vai ter a permissão de gravar em log qualquer acontecimento anormal em termos de comunicação do sistema. (Na falta de resposta dos sensores, receção de mensagens do tipo ERROR.)
16. Em termos do modulo de processamento, ele tem a capacidade de:
  - Verificar os valores das amostras;
  - Identificar valores com uma probabilidade elevada de terem ERROS;
  - Detetar eventuais situações anómalas;
  - Todos os valores que resulta de uma análise protocolar 100 % correta das mensagens que vão ser recebidas no Arduíno têm de ser gravados no ficheiro respetivo num formato de texto ou CSV.
17. A entrada/linha no ficheiro de amostras em formato de CSV vai ter a seguinte sintaxe:

ISS; ISu; etiqueta\_temporal; valor\_PA; num\_amostra; valor\_amostra

## Comunicação do Sistema

1. Os concentradores terão de ter uma interface simples e um fluxo exterior de dados limitados a ficheiros de configuração e ficheiros de log (out).
2. O protocolo de comunicação entre o concentrador e o sensor na fase é o *PDUs* (protocolo transporte *UDP/IP*, componentes simuladas ou sistema equipamento de rede sem-fios).
3. No caso em que utilizamos sistemas de sensores simulados, o protocolo tem de ser implementado uma interface *socket UDP/IP* entre o concentrador e o Arduino. Os *PDUs* devem permitir a definição de vários tipos de mensagem, sendo algumas das mensagens de tamanho variável.

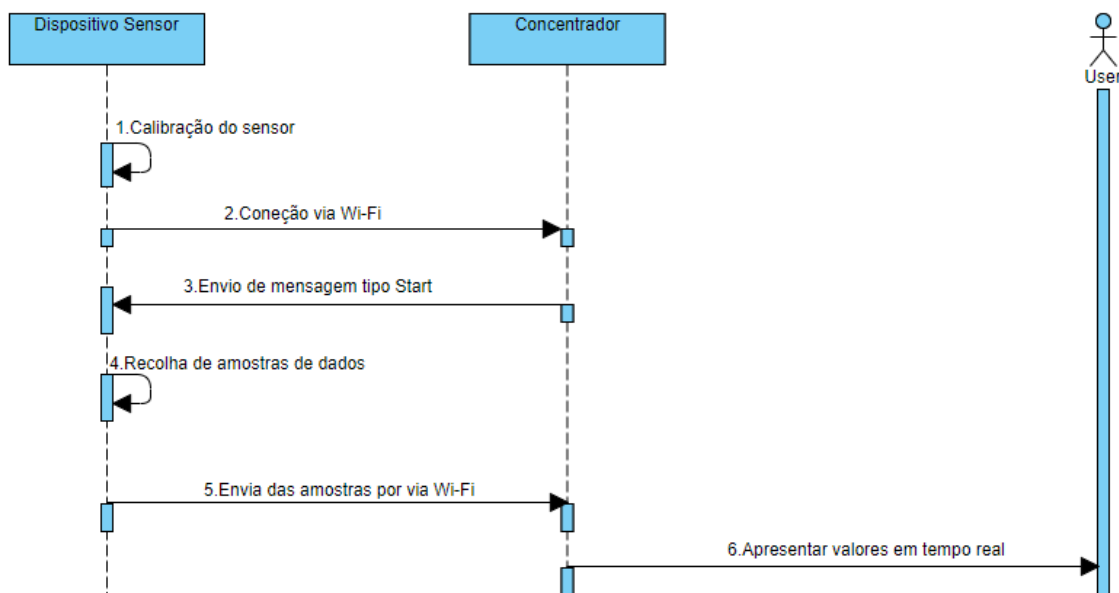


Figura 8 - Diagrama de Sequência, da comunicação entre sistema sensor e concentrador

# Arquitetura

Nesta primeira fase do projeto tivemos como objetivo o desenvolvimento do sistema sensor conforme demonstrado na Figura 8.

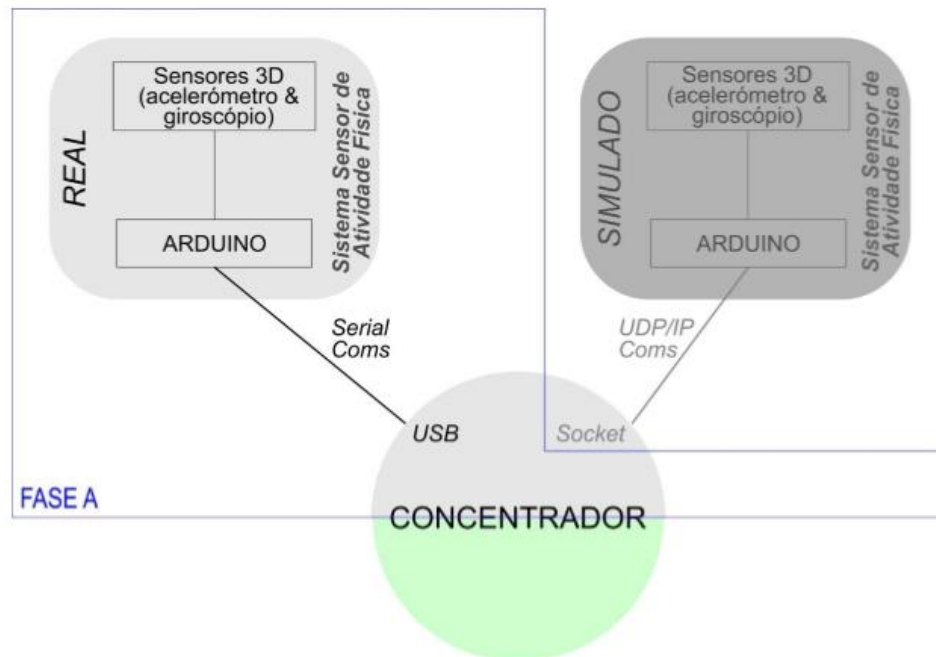


Figura 9 - Sistema sensor de Atividade Física

Na figura 8, podemos visualizar uma imagem que possibilita ilustrar o sistema requerido para a Fase A do projeto. Através desta imagem é possível refletir as soluções que nos permite alcançar a um ponto final do projeto.

Resumidamente, nesta Fase A tivemos como principal objetivo o desenvolvimento de hardware e de software necessário para a recolha de dados por parte dos sensores.

A implementação do circuito apenas inclui ligações físicas entre componentes (MPU 6050) e o controlador (ESP32-WROOM-32D). Na Figura 9 é possível visualizar o esquema da montagem realizada pelo grupo de uma forma organizada e na tabela 5 conseguimos ver a relação entre módulos e cores do esquema da Figura 9.

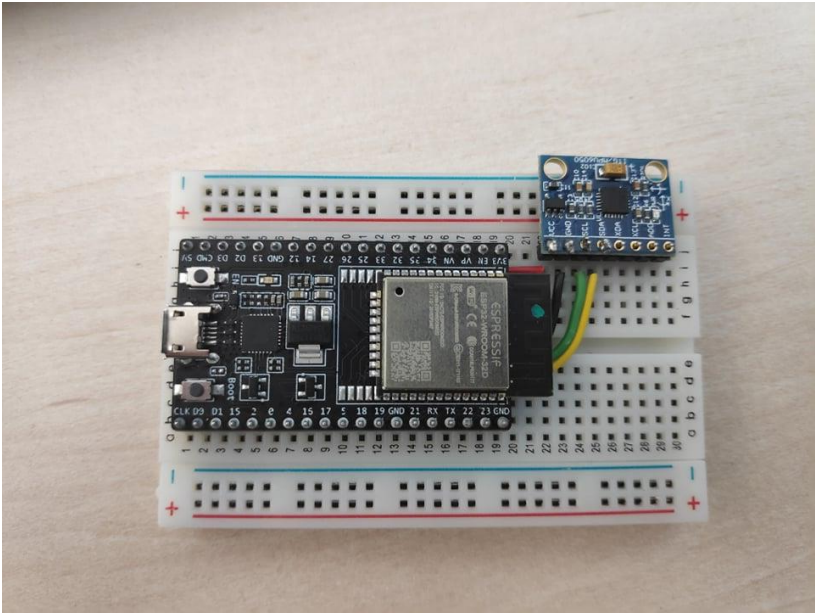


Figura 10 - Sistema Sensor

Cor (Fios)	ESP32-WROOM-32D (Porta)	Modulo MPU6050 (Porta)
Vermelho	3V3	VCC
Preto	GND	GND
Verde	22	SCL
Amarelo	21	SDA

Tabela 5 - Ligações entre ESP32 e modulo MPU6050

## Arquitetura do software Arduino-Sensor

Aqui, vamos referir o raciocínio própria de toda a componente de software que esta aplicável no sistema, explicando as opções tomadas ao longo da construção do mesmo e também o modo implementação.

No código inerente ao Arduino, este é responsável por as seguintes tarefas:

- Recolher amostras efetuadas pelo sensor
- Utilização do protocolo de comunicação UDP
- Transmissão com o concentrador

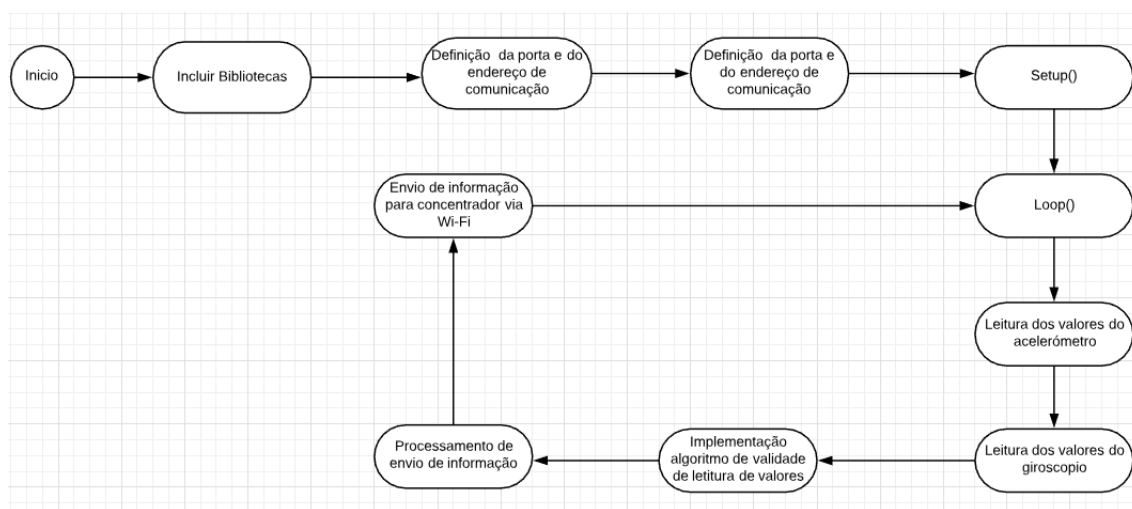


Figura 11 - Algoritmo do código no Arduino: processamento de recolha/envio de informação

Para a realização deste código foi necessária incluir algumas livrarias para um melhor funcionamento do nosso sistema, estas livrarias são referenciadas na secção mais em baixo [1].

O nosso programa começa logo de início com uma calibração correta do sensor, para isso incluímos uma livraria para termos uma recolha de amostras muito mais precisa e correta. Tendo uma calibração correta no sensor, de seguida o Arduino vai realizar a conceção por via Wi-Fi utilizando um protocolo UDP. Na seguinte figura podemos ver as atribuições da porta, endereço e das bibliotecas que foram incluídas.



```
#include "Arduino.h"
#include "MPU6050_tockn.h"
#include "Wire.h"
#include "I2Cdev.h"
#include "TimeLib.h"

#include <WiFi.h>
#include <WiFiUdp.h>
```

Figura 12 - Bibliotecas usadas no Arduino

De seguida na parte do Setup(), seleccionamos um *baudrate* que pretendemos utilizar no sistema e implementamos todo o código que foi necessário para realizar uma comunicação via Wi-Fi e também estabelecemos uma rota. A função Setup(), é sempre executada apenas uma vez, após cada energização ou reinício da placa Arduino.

```
void setup() {
  Serial.begin(115200);
  Wire.begin();
  while (!Serial) ; // wait for serial port to connect. Needed for native USB

  //attempt to connect to Wifi network:
  while(WiFi.status() != WL_CONNECTED){
    Serial.println("Trying to connect.");
    WiFi.begin(ssid, password);
    Serial.println("Please wait...");
    delay(10000);
  }
  Serial.print("WiFi Connected.");
  udp.begin(udpPort);
  mpu6050.begin();
  mpu6050.calcGyroOffsets(true);

  //estabelecer rota
  udp.beginPacket(udpAddress,udpPort);
  sprintf((char *) buffer,"-l|%d",iSS);
  udp.write(buffer,sizeof(buffer));
  udp.endPacket();
  //fim de envio de data
}
```

Figura 13 - Estabelecimento de conexão WIFI, ligação à porta através do protocolo UDP e envio da trama READY

Na função loop() como o próprio nome indica, realiza ciclos consecutivamente, permitindo que o nosso programa seja alterado e respondido. Usamos esta secção para controlar ativamente a placa do Arduino.

Nesta função temos o código onde faz o controlo da informação vinda do concentrador. Está implementado no Arduino a receção de informação do tipo START, STOP ou ERROR como podemos visualizar nas figuras 15, 16 e 17.





```

void loop() {
    memset(buffer, '\0', BUFFERSIZE); //limpeza da variavel
    //verificacao se existe algum pacote
    int packetSize=udp.parsePacket();
    errorPacket=false;
    if (packetSize) {
        while((udp.read(buffer, BUFFERSIZE))>0){}
        buffer[packetSize+1]='\0';
        //Serial.println((char *) buffer);
        if (buffer[0]=='0') {
            /* Start */
            startData=true;
            start();
        }else if (buffer[0]=='2') {
            /* stop */
            startData=false;
        }
    }
    if (startData || errorPacket) { //se nao e se ja recebeu start
        if(startData){
            data();
        }
        //processo de envio de pacote
        udp.beginPacket(udpAddress,udpPort);
        udp.write((uint8_t *) buffer,sizeof(buffer));
        udp.endPacket();
        memset(buffer, '\0', BUFFERSIZE);
        //fim de envio de data
    }
}
}

```

Figura 14 – Código do Arduino com leitura de trama e envio

Como podemos visualizar, dentro da função loop() nos chamamos outras funções para um melhor funcionamento do nosso código. Estas funções nos demos de nome de start(), data() e error(). Na seguinte figura podemos visualizar o código das respetivas funções que foram referenciadas em cima.



```

void start() { //LEITURA
    int h,m,s,d,mth,y;
    Serial.println((char *)buffer);
    sscanf((char *) buffer, "%d|%d|%d|%d:%d-%d/%d/%d", &startS.TP, &startS.PM, &startS.PA, &startS.NS, &h, &m, &s, &d, &mth, &y);
    setTime(h,m,s,d,mth,y);
    sprintf(startS.TS, "%d:%d:%d-%d-%d/%d/%d", h,m,s,d,mth,y);

    if(startS.NS>6){
        error(0); //ns superior ao limite
    }
    if(!errorPacket){
        if(startS.PA<pa){
            error(1); //limite de colheita de dados
        }
    }
    if(!errorPacket){
        if(startS.PM<=(startS.PA*startS.NS)){
            error(2); //tamnho pm insuficiente
        }
    }
}
}

```

Figura 15 - Função START, com interpretação da trama e procura de erros

```

void data() { //ENVIO
    int timerPA, timerPM=millis();
    char amostra[20];
    memset(amostra, '\0', 20);
    memset(dataS.amostra, '\0', 128);
    dataS.TP=1;
    dataS.ISS=ISS;
    sprintf(dataS.TS, "%d:%d:%d-%d-%d/%d/%d", hour(), minute(), second(), day(), month(), year());
    if (startS.PM==0) { dataS.PM=pm; } dataS.PM=startS.PM;
    if (startS.PA==0) { dataS.PA=pa; } dataS.PA=startS.PA;
    if (startS.NS==0) { dataS.NS=ns; } dataS.NS=startS.NS;

    for (int i = 0; i < dataS.NS; i++) {
        //iniciar limite de pa
        timerPA=millis();
        mpu6050.update();
        sprintf(amostra, "<%2f|%.2f|%.2f>", constrain(mpu6050.getAccX(), -1, 1), constrain(mpu6050.getAccY(), -1, 1), constrain(mpu6050.getAccZ(), -1, 1));
        strcat(dataS.amostra, amostra);
        while((timerPA+dataS.PA)<millis()){
            //erro
            error(3); //erro de timeout de pa
        }
    }
    Serial.println(dataS.amostra);
    sprintf((char *) buffer, "%d|%d|%d|%d|%s|%s", dataS.TP, dataS.ISS, dataS.PM, dataS.PA, dataS.NS, dataS.TS, dataS.amostra);
    while((timerPM+dataS.PM)<millis()){
        error(4); //erro de timeout de pm
        Serial.println("PM Erro");
    }
}
}

```

Figura 16 - Função DATA



```
void error(int tipo){
    errorPacket=true;
    startData=false;
    errorS.TP=3;
    errorS.ISS=iSS;
    sprintf(errorS.TS, "%d:%d:%d-%d/%d/%d", hour(), minute(), second(), day(), month(), year());
    errorS.ER=tipo;
    sprintf((char *) buffer, "%d|%d|%s|%d", errorS.TP, errorS.ISS, errorS.TS, errorS.ER);
}
```

Figura 17 - Função ERROR

Para explicar como implementamos o nosso protocolo de camada de transporte, decidimos detalhar todo o processo que foi necessário para o realizar. Logo de início temos de incluir a biblioteca WifiUDP.h para ser possível realizar este processo de comunicação.

A biblioteca WifiUDP.h é necessária para programar as rotinas UDP. Tendo as bibliotecas abrangidas, podemos criar um objeto WifiUDP. Para isso é necessário especificar uma certa porta para receber os pacotes e de seguida temos de configurar o buffer para pacotes de entrada e definir uma mensagem de resposta.

```
const char * ssid = "hugodocpu";
const char * password = "DV9SrOap";

const char * udpAddress = "10.42.0.1";
const int udpPort = 11111;
```

Figura 18 – Dados de configuração do access point

Para realizar a comunicação Wi-Fi, usamos o nosso computador como um *access point*, incluindo as configurações no código do Arduino.

Através do Wifi.Begin(), onde inicia as configurações de rede da biblioteca Wi-Fi e também o seu status atual. No nosso caso nos utilizamos a seguinte sintaxe: Wifi.begin(ssid, pass). O parâmetro ssid (Service Set Identifier) corresponde ao nome da rede Wi-Fi na qual nos desejamos conectar, sendo no nosso caso denominado por "hugodocpu". O parâmetro pass corresponde à senha que vamos utilizar como sistema de segurança e no nosso programa esta configurada como "DV9SrOap".

Através deste método, é possível receber os seguintes *returns*:

- WL\_CONNECTED, quando estamos conectados na rede
- WL\_IDLE\_STATUS, quando não estamos conectados na rede



Tendo a conexão estabelecida de forma correta, podemos iniciar a ligação WIFI à porta escolhida através do protocolo UDP.

```
udp.begin(udpPort);
```

**Figura 19 – Conexão à porta através do protocolo UDP**

Através deste modelo de código, é possível enviar e receber pacotes UDP entre o ESP32 e um aplicativo externo.

## Arquitetura do software Concentrador

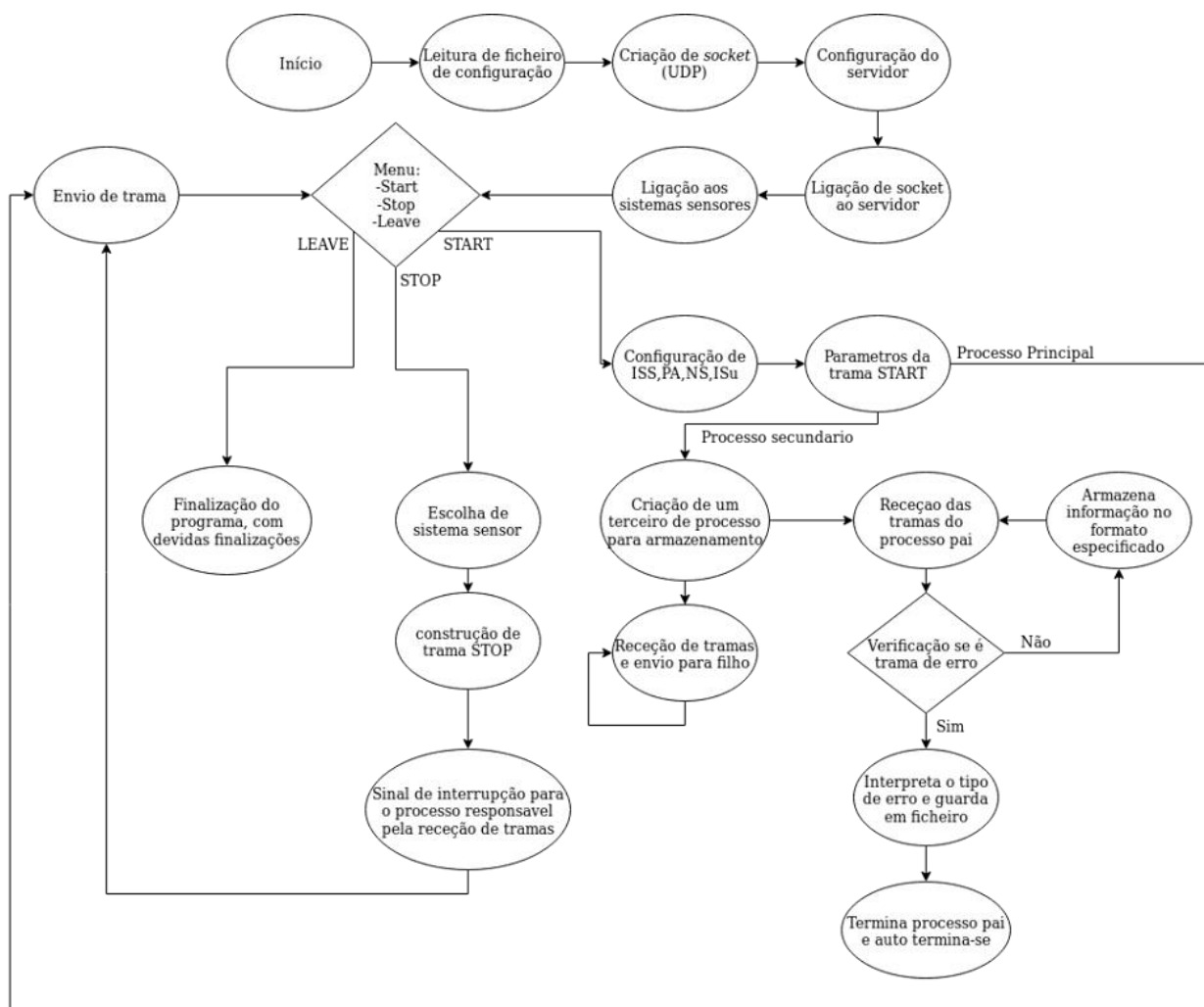


Figura 20 - Diagrama de blocos do concentrador (em linguagem natural)

Tudo começa através da execução do concentrador, um processo que integra a leitura de um ficheiro configuração do nosso sistema, o qual que contem número da porta, do PA e do valor NS.

Após a leitura do ficheiro de configuração, de seguida é criado um ficheiro de “setup.txt” que irá servir para o registo das várias etapas de configuração do sistema:

- Criação de um *socket* com o protocolo UDP (definido no enunciado da fase A);
- Configuração do servidor (atribuição de IP e Porta);
- Ligação do servidor à *socket*.



Tendo as configurações de comunicação todas estabelecidas e inseridas, conseguimos visualizar um menu no qual é realizada a conexão dos sistemas sensor. Neste procedimento, aguardamos pela conexão dos sistemas sensores, onde implementamos uma trama READY com o objetivo de identificar o sistema sensor, onde avisa o concentrador que o sistema sensor que se encontra completamente calibrado e pronto para ser utilizado, também é armazenado o IP do sistema sensor juntamente com a sua identificação.

Feita a conexão com sucesso e todos os parâmetros de configuração dos sistemas sensores existentes, é realizado o processo de controlo dos mesmos. No processo de controlo teremos 3 opções disponíveis:

1. Inicia o processo da trama START;
2. Inicia o processo da trama STOP;
3. Saída do concentrador.

Na primeira opção, é-nos apresentada a escolha do sistema sensor que pretendemos usar, de seguida será pedida a identificação do sujeito e por fim é possível usar os dados do ficheiro de configuração ou introduzimos nós próprios os parâmetros de PA e NS.

Estando a inserção de valores realizada com sucesso, é inicializado um novo processo para receber as tramas DATA do sistema sensor enquanto que o processo principal trata de enviar a trama START, voltando para o menu. No processo secundário antes de iniciar receção das tramas é criado um outro processo para o armazenamento de amostras feitas em *log\_files*.

A comunicação entres estes dois processos secundários que nós acabamos de descrever, é feita através de *pipes* anónimos. Aqui incluímos (um extra) um *pipe* com nome, conhecido também como *fifo*, de forma a ser possível fazer uma monitorização das amostras (em tempo real) numa janela separada. Caso seja recebida alguma trama ERROR, é realizado a criação de um ficheiro de erro contendo a explicação do mesmo terminando de seguida os processos secundários.

Por último, na segunda opção, escolhemos o sistema sensor que pretendemos terminar. Antes de efetuar o envio da trama STOP para o sistema sensor é enviado um sinal de interrupção (*SIGINT*) para o processo responsável pela receção das tramas que faz com que termine de uma forma segura e desejável, e consequentemente o processo que efetua o armazenamento é também terminado.



## Bibliotecas utilizadas

Nesta secção vamos falar sobre as bibliotecas que foram utilizadas para realizar o projeto que nos foi proposto.

```
#include "Arduino.h"
#include "MPU6050_tockn.h"
#include "Wire.h"
#include "I2Cdev.h"
#include "TimeLib.h"

#include <WiFi.h>
```

Figura 21 – Bibliotecas usadas no código arduino

A biblioteca de dispositivos I2C é uma coleção de classes uniformes e com uma boa documentação para fornecer interfaces simples e intuitivas para dispositivos I2C. Cada dispositivo é contruído para usufruir da classe I2Cdev genérica, que abstrai a comunicação de nível de bit e de byte de cada classe de dispositivo específico, mantendo a classe de dispositivo limpa tornando mais simples de modificar.

Também foi necessário usar a biblioteca Wire.h, contendo duas *wire interfaces* (TWI/I2C) para ser possível receber e enviar dados através de uma rede de dispositivos ou sensores. A implementação desta biblioteca usa um buffer de 32 bytes, portanto qualquer comunicação que seja aplicável com esta biblioteca deve se encontrar dentro desse limite. Excedendo esse limite a comunicação é descartada.

No início de qualquer código é obrigatório incluir a biblioteca wire.h. Após esta biblioteca ter sido incluída, é necessário iniciar dentro de setup() com Wire.Begin(). Através do Wire.begin(), inicia a biblioteca Wire e junta-se com o I2C como *master* ou *slave*.

A biblioteca MPU6050\_tockn.h é uma biblioteca para tornar uma comunicação mais fácil e acessível com o sensor MPU6050. Para obter dados do sensor MPU6050, é necessário primeiro executar o método update(). Este método irá obter todos os dados do sensor, sendo possível calcular o angulo pelo acelerómetro e giroscópio.

```
mpu6050.update();
sprintf(amostra, "<%.2f|%.2f|%.2f>",
        constrain(mpu6050.getAccX(), -1, 1),
        constrain(mpu6050.getAccY(), -1, 1),
        constrain(mpu6050.getAccZ(), -1, 1));
```

Figura 22 - Método usado para obter valores dos eixos do acelerómetro



Esta biblioteca também inclui um processo de auto calibração por parte do sensor MPU6050. Incluímos no nosso código o método `calcGyroOffsets()` no `setup()`, esse método vai calcular a calibração do giroscópio. Através do `calcGyroOffsets(true)` podemos visualizar o estado de calibração do sensor através do monitor.

```
Wire.begin();  
mpu6050.begin();  
mpu6050.calcGyroOffsets(true);
```

Figura 23 - calibração giroscópio

Inclui-se também a biblioteca `time.h`, uma biblioteca que fornece a funcionalidade de cronometragem para o Arduino. Esta biblioteca integra as seguintes funções:

- `hour();`
- `minute();`
- `second();`
- `day();`
- `weekday();`
- `month();`
- `year();`

Para realizar uma comunicação via Wi-Fi, foi necessário implementar a biblioteca `WiFi.h`. Através desta biblioteca, o Arduino pode servir como um servidor aceitando conexões de entrada ou um cliente fazendo conexões externas.

Uma das bibliotecas que também foi utilizada foi a `WiFiUdp.h`, biblioteca que permite enviar e receber mensagens UDP. Esta biblioteca é necessária para a programação de rotinas UDP. Após incluir esta biblioteca é necessário criar um objeto `WiFiUDP`, onde devemos especificar a porta de leitura de pacotes.



# Estrutura do protocolo de dados

Através de uma análise do enunciado do projeto, conseguimos desenvolver um protocolo de comunicação assíncrono.

Através da implementação do protocolo vai ser possível a realização de troca de dados entre o Arduino e o Concentrador. Para começar vai ser enviada uma trama *START*, que vai ser enviada através do concentrador. O *START* vai ser constituído pelos parâmetros que vai estabelecer a comunicação do sistema, constituído por 28 *bytes* de modo a reservar todos os parâmetros necessários.

Através desses parâmetros, vai ser possível o Arduino enviar uma trama *DATA* onde vai alocar valores das amostras recolhidas.

Na trama *DATA*, o tamanho da trama é instituído dependendo das dimensões dos critérios de configuração e também depende do número de amostras que é pretendido enviar. Para cada amostra são necessários 19 *bytes* para armazenamento de informação.

Recebendo uma mensagem de deteção de ERRO, vai ser enviado uma mensagem para o concentrador o tipo de erro que sucedeu durante a receção de dados.

Para a explicação do esquema de comunicação entre o Concentrador e Arduino, recorremos na figura 4. Também temos disponível uma legenda da figura, dispondo todos os parâmetros que vai ser necessários usar nas diferentes tramas utilizadas.

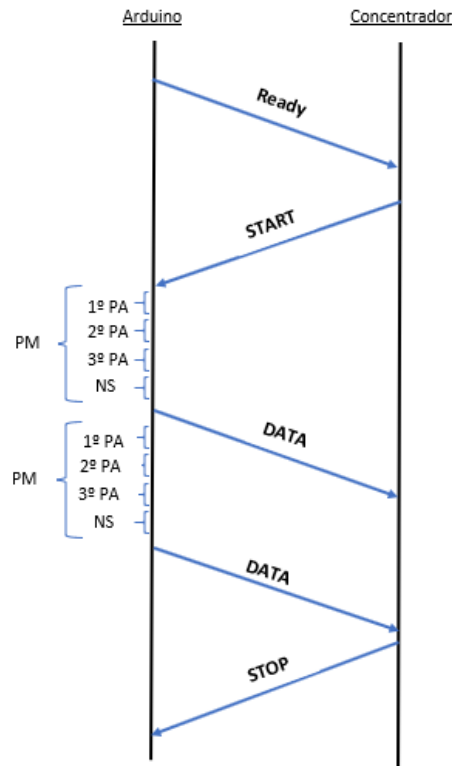


Figura 24 - Funcionamento do Protocolo



## Tipos de Mensagens

### 1. DATA

Mensagem que é enviada do Arduino para o concentrador com os respetivos valores de amostra. Parâmetros da mensagem:

- TP - Tipo de mensagem;
- ISS - Identificação do Sistema Sensor;
- TS - *timestamp* do momento em que recolhe o valor da primeira amostra que vais estar presente na mensagem;
- PM / PA - Valores de PM e PA para a recolha de amostras que vão estar presentes na mensagem;
- NS - número de amostras presentes na Trama;
- Amostras (S1, S2, ...) – número mínimo de amostras 1 e máximo 6.

Temos de ter uma grande atenção na estrutura da trama para possibilitar uma correta interpretação dos valores que vamos receber.

#### Estrutura:

Byte	1	1	19	4	3	1	19	...	Total
Parâmetro	TP (1)	ISS	TS	PM	PA	NS	S1	S.ns	143/48 Bytes

### 2. ERROR

Mensagem que vai ser enviada do Arduino para o concentrar para alertar ao utilizador que existe algum erro no sistema. Parâmetros da Mensagem:

- TP - Tipo de mensagem;
- ISS - Endereço do sistema sensor;
- TS - *timestamp* do momento em que ocorre o erro e o valor do tipo de erro.
- ER:
  - 0-Erro no número de amostras;
  - 1-Erro no período de amostragem;
  - 2-Erro no período de mensagem;
  - 3-Ultrapassou período de amostragem;
  - 4-Ultrapassou período de mensagem.

#### Estrutura:

Byte	1	1	19	1	Total
Parâmetro	TP (3)	ISS	TS	ER	2 Bytes



### 3. START

Mensagem que é enviada do Concentrador para o Arduino, a pedir autorização de início de recolha e envio de amostras. Parâmetros da Mensagem:

- TP - Tipo de Mensagem;
- ISS - Endereço do sistema sensor;
- TS - *timestamp* do momento em que envia a mensagem;
- PM – Período de mensagem máximo;
- PA – Período de amostragem mínimo;
- NS – Número de amostras.

#### Estrutura:

Byte	1	1	19	4	3	1	Total
Parâmetro	TP (0)	ISS	TS	PM	PA	NS	28 Bytes

### 4. STOP

Mensagem que é enviada do concentrador para o Arduino para pedir a finalização de recolher ou envio de dados. Parâmetros da Mensagem:

- TP - Tipo de mensagem;
- SR - valor indicativo da razão de paragem;

#### Estrutura:

Byte	1	1	Total
Parâmetro	TP (2)	SR	2 Bytes

### 5. READY

Mensagem que é enviada do Arduino para o concentrador a avisar que o sistema sensor encontra-se disponível para ser utilizado. Parâmetros da Mensagem:

- TP - Tipo de mensagem;
- ISS - Endereço do sistema sensor;

#### Estrutura:

Byte	2	1	Total
Parâmetro	TP (-1)	ISS	4 Bytes



## Formato dos valores recolhidos

Para formatação de valores nos utilizamos um formato do tipo ficheiro CSV para uma melhor organização das amostras recolhidas.

É um ficheiro sem formatação em que os valores vão se encontrar separados por vírgulas, delimitados por aspas e em cada linha vai ter um registo diferente, ou seja, um cliente, fornecedor ou um artigo por linha.

Em baixo encontrasse um exemplo de uma linha de um ficheiro CSV que deverá ter a seguinte aparência quando vamos visualizar num bloco de notas ou outro programa que seja do mesmo género.

```
"Pera","Origem:Portugal","889","30","1.10","1".
```

Este tipo de ficheiro pode ser criado em qualquer tipo de software de Folha de Cálculo, como por exemplo Microsoft Excel, OpenOffice. Para a criação destes tipos de ficheiros nestes programas que demos como exemplo, é necessário que em cada linha tenha uma correspondência a um registo diferente, colocando sempre em cada valor numa célula diferente.

Para gerir um ficheiro de CSV a partir de um ficheiro.txt, teremos que manter a estrutura original do Ficheiro CSV, isto é, todos os valores têm de ser delimitados por aspas ( " ") e separados por vírgulas. A partir disto, deverá guardar o ficheiro em formato CSV.



# Testes e análises de resultados

---

Nesta secção vamos expor a avaliação do desempenho do sistema que nos foi proposto implementar no qual será feita uma análise critica aos resultados que nos conseguimos obter.

Nesta secção vamos utilizar apenas valores que nos recolhemos através do sensor e uma aplicação onde nos fornece os dados corretos da posição que o sensor se encontra. A aplicação que nos utilizamos para recolhermos os valores corretos para fazer o teste ao nosso sistema sensor foi SensorTest. Esta aplicação fornece várias ferramentas, mas o que é essencial para fazer testes ao nosso sistema é a ferramenta de Acelerómetro e Giroscópio.

Característica do Sensor Teste Acelerómetro:

- Nome: BMI120 Accelerometer
- Vendedor: Bosh
- Max range: 156.906477
- Unit: m/s<sup>2</sup>

Característica do Sensor Teste Giroscópio:

- Nome: BMI120 Gyroscope
- Vendedor: Bosh
- Max Range: 34.906586
- Unit: rad/s

Um dos grandes objetivos deste projeto é obter uma leitura de dados o mais preciso possível e avaliar o erro que obtivemos em relação aos dados recolhidos através do sensor.

Para determinar o erro absoluto da nossa recolha de dados, é necessário avaliar a proximidade do valor lido com o valor recolhido:

$$ErroAbsoluto = X - x$$

X - Valor exato da medida efetuada.

x - Valor que foi recolhido pelo nosso sistema sensor.



Mas não é só através do erro absoluto que nos conseguimos concluir se o nosso valor recolhido se encontra o mais correto possível, é necessária também determinar o erro relativo dependendo do erro absoluto:

$$ErroRelativo = \frac{X - x}{x}$$

X- Valor exato da medida efetuada.

x- Valor que foi recolhido pelo nosso sistema sensor.

O sistema sensor foi testado em diferentes posições, testando várias posições do sistema através do sistema de eixos X, Y, Z.

Na tabela X é apresentado os valores lidos pelo sistema sensor (Acelerómetro + Giroscópio) e podemos verificar que variando a posição do sistema sensor varia adequadamente e por isso o sistema sensor permite recolher valores em várias posições. Com base na aplicação de teste de Sensor para comparar a leitura dos dados que foram obtidos podemos concluir que o sistema sensor apresenta valores fiáveis, precisos e exatos.

Número da Amostra:	Valor Recolhido:	Valor Exato:	Erro Absoluto:	Erro Relativo:
Horizontal	X: 0.04 Y: 0.01 Z: 1.00	X: 0 Y: 0 Z: 1.00	X: -0.04 Y: -0.01 Z: 0.00	X: -1 Y: -1 Z: 0
A	X: 1.00 Y: 0.00 Z: 0.002	X: 0.98 Y: 0.002 Z: 0.001	X: -0.02 Y: 0.002 Z: -0.001	X: -0.02 Y: -0.002 Z: -0.5
Amostra 3	X: 0.02 Y: 1 Z: 0.01	X: 0.001 Y: 0.99 Z: 0.008	X: -0.019 Y: -0.01 Z: -0.002	X: -0.95 Y: -0.01 Z: -0.2

Tabela 6 - Valores obtidos a partir do sensor Acelerómetro



# Conclusão

---

Concluída esta fase, apesar das várias dificuldades, conseguimos contornar e obter vários conhecimentos a nível de programação de Arduino como também em linguagem C, alcançando os objetivos propostos na Fase A.

Conseguimos comunicar um ou mais sistemas sensores ao concentrador através de tecnologia sem fios (Wi-Fi) usando o protocolo UDP, proposto como requisito opcional. Contudo ao longo da fase encontramos várias dificuldades e mesmo obtendo o funcionamento pedido, houveram adversidades que nos impediram de realizar o projeto que ambicionamos, ficando em falta a inclusão dos dados do giroscópio e temperatura por amostra e um erro relacionado com as tramas ERROR.

Em suma, esta fase permitiu-nos testar os nossos conhecimentos e ainda adquirir outros. Pretendemos na próxima fase continuar a adquirir ainda mais conhecimentos e aplicar os obtidos até ao momento, possibilitando superar todas as adversidades que obtivemos nesta fase.



## Referencias bibliográficas

---

UDP Server-Client implementation in C. Disponível em:  
<<https://www.geeksforgeeks.org/udp-server-client-implementation-c/>>.  
Acesso em: 14 fev. 2019.

WiFi library. Disponível em:  
<<https://www.arduino.cc/en/Reference/WiFi>>.  
Acesso em: 20 fev. 2019.

Arduino core for the ESP32. Disponível em:  
<<https://github.com/espressif/arduino-esp32>>.  
Acesso em: 26 fev. 2019.

Arduino library for easy communicating with the MPU6050. Disponível em:  
<[https://github.com/Tockn/MPU6050\\_tockn](https://github.com/Tockn/MPU6050_tockn)>.  
Acesso em: 1 mar. 2019.

Como ligar o MPU6050 ao ESP32. Disponível em:  
<<https://www.circuito.io/app?components=9442,11028,360217>>.  
Acesso em: 2 mar. 2019.