ISEL
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Licenciatura em Engenharia Informática e Multimédia (LEIM)

Computação Física – 1819SV

Trabalho prático nº3

Engº Carlos Carvalho

Turma LEIM23D

Luís Fonseca	nº 45125
Tiago Oliveira	nº 45144
Rodrigo Correia	nº 45155

14/06/2019

Índice

Índice Figuras e Tabelas

Objetivos/Introdução

A realização deste projeto tinha por objetivo o estudo de um sensor, contendo um giroscópio com três eixos (L3GD20) e um display, ambos com protocolo de comunicação I2C. Para o estudo do sensor foi disponibilizado o "*data sheet"* que contém todas as informações necessárias para utilizar o mesmo de forma correta. Tanto a informação acerca do display, como do sensor encontrava-se disponibilizada nas folhas dos Engenheiros.

Retirada a informação necessária, foi feita uma implementação no Arduino, no qual foi calculado o valor dos três eixos e da temperatura. Com estes passos concluídos, procedeu-se ao envio destes valores para o display.

Por último foi feito um programa em *Python*, no qual era apresentado o valor da velocidade angular. Para isso foi feita a ligação do Arduíno ao *python*, e através do *pygame*, representamos então os valores que eram pretendidos.

Na sequência deste trabalho definiu os seguintes objectivos que permitiram concretizar este projecto:

- Ler o datasheet acerca do sensor;
- Ler e interpretar as funções que eram pedidas para a construção do display;
- Depois de ser feito o envio de frases para eventuais testes no display, ler e interpretar a informação acerca do sensor e de como calcular os três eixos e a temperatura;
- Depois das funções acerca do sensor estarem feitas, passou-se para a construção de autómatos, no qual irá servir para representar a informação acerca dos valores dos eixos no display, e o valor da velocidade angular no *python*, ambos partilham a informação acerca da temperatura;
- Depois dos autómatos feitos, passou-se para a construção de um programa em *Python*, para representar a velocidade angular e a temperatura;

Desenvolvimento

Sensor L3GD20

O sensor L3GD20 a estudar, tem uma interface de saída I2C. Por sua vez, este sensor tem como funcionalidade o cálculo da temperatura (ºC) e um giroscópio, no qual, os cálculos são efetuados à custa dos três eixos, X,Y e Z.

Internamente, o dispositivo é mantido a vibrar a uma frequência conhecida, e, através da perturbação dessa vibração, pode estimar-se a velocidade angular sob cada um dos três eixos X, Y e Z.

Esta informação é fornecida considerando-se a rotação no sentido oposto ao dos ponteiros do relógio, estando de frente para o respetivo eixo considerado, obtendo-se ΩX, ΩY e ΩZ. A sensibilidade é variável, de acordo com três fins de escala.

Para a obtenção dos resultados, é necessário seguir uma série de procedimentos disponibilizados no"*data sheet"* fornecido pelo fabricante.

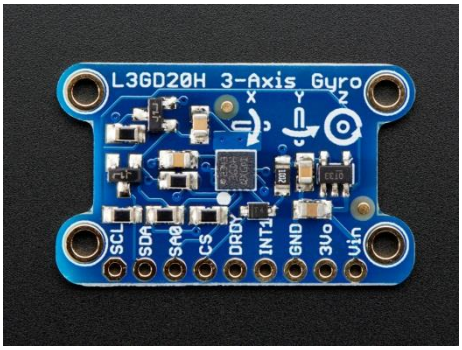

Figura 1 - sensor L3GD20

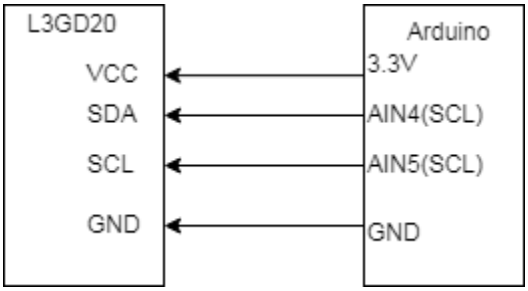

Figura 2 - ligação entre o sensor e o Arduino

Protocolo I2C

O protocolo I2C é baseado em dois sinais digitais, o SDA (*Serial Data*) e o SCL (*Serial Clock*) e tem como função a comunicação entre dispositivos. SDA é o pino que efetua a transferência de dados, e SCL tem como função a temporização entre os dispositivos. O protocolo tem uma condição de *start*, na qual há uma transição descendente em SDA enquanto o SCL tem o valor de 1, também este possui uma condição de stop que ocorre quando há uma transição ascendente do sinal SDA estando o sinal SCL com o valor de 1.

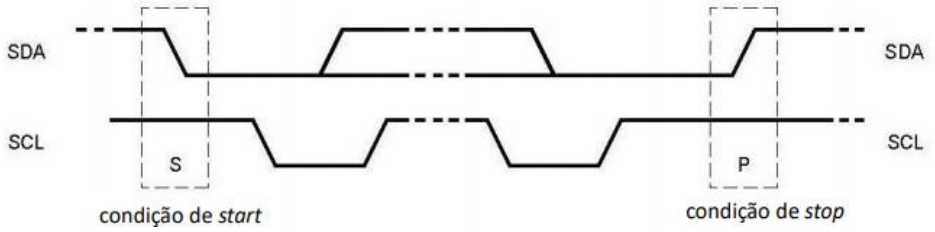

Figura 3 -Condições de start e stop do protocolo I2C

A tabela a seguir, mostra os endereços que foram usados para representar a temperatura e os diferentes valores dos três eixos do giroscópio.

Name	Type	Register address		Default
		Hex	Binary	
Reserved	-	00-0E	-	-
WHO_AM_I	r	0F	000 1111	11010100
Reserved	-	10-1F	-	-
CTRL_REG1	rw	20	010 0000	00000111
CTRL_REG2	rw	21	010 0001	00000000
CTRL_REG3	rw	22	010 0010	00000000
CTRL_REG4	rw	23	010 0011	00000000
CTRL_REG5	rw	24	010 0100	00000000
REFERENCE	rw	25	010 0101	00000000
OUT_TEMP	r	26	010 0110	output
STATUS_REG	r	27	010 0111	output
OUT_X_L	r	28	010 1000	output
OUT_X_H	r	29	010 1001	output
OUT_Y_L	r	2A	010 1010	output
OUT_Y_H	r	2B	010 1011	output
OUT_Z_L	r	2C	010 1100	output
OUT_Z_H	r	2D	010 1101	output
FIFO_CTRL_REG	rw	2E	010 1110	00000000
FIFO_SRC_REG	r	2F	010 1111	output
INT1_CFG	rw	30	011 0000	00000000
INT1_SRC	r	31	011 0001	output
INT1_TSH_XH	rw	32	011 0010	00000000
INT1_TSH_XL	rw	33	011 0011	00000000
INT1_TSH_YH	rw	34	011 0100	00000000
INT1_TSH_YL	rw	35	011 0101	00000000
INT1_TSH_ZH	rw	36	011 0110	00000000
INT1_TSH_ZL	rw	37	011 0111	00000000
INT1_DURATION	rw	38	011 1000	00000000

Tabela 1 - endereços usados para a obtenção da temperatura e dos eixos do giroscópio

Passando agora para a explicação acerca dos endereços usados:

- WHO_AM_I – corresponde à identificação do sensor, com o endereço, 0xD3 ou 0xD3, sendo estes endereços, endereços por default;

WHO_AM_I	r	0F	000 1111	11010100

Figura 4 - endereço do parâmetro WHO_AM_I

- OUT_X_H e OUT_X_L – Informação a oito bits sobre a parte alta e a parte baixa do valor digitalizado da velocidade angular sobre o eixo do X.

OUT_X_L	r	28	010 1000	output
OUT_X_H	r	29	010 1001	output

Figura 5 - endereços dos parâmetros X_LOW e X_HIGH usados

- OUT_Y_H e OUT_Y_L – Informação a oito bits sobre a parte alta e a parte baixa do valor digitalizado da velocidade angular sobre o eixo do Y.

OUT_Y_L	r	2A	010 1010	output
OUT_Y_H	r	2B	010 1011	output

Figura 6 - endereços dos parâmetros Y_LOW e Y_HIGH usados

- OUT_Z_H e OUT_Z_L – Informação a oito bits sobre a parte alta e a parte baixa do valor digitalizado da velocidade angular sobre o eixo do Z.

OUT_Z_L	r	2C	010 1100	output
OUT_Z_H	r	2D	010 1101	output

Figura 7 - endereços dos parâmetros Z_LOW e Z_HIGH usados

- OUT_TEMP – Informação a oito bits, expressa em complemento para dois, sobre a temperatura do ambiente onde o sensor está colocado.

OUT_TEMP	r	26	010 0110	output

Figura 8 - endereços do parâmetro OUT_TEMP usado

Se a velocidade angular for constante, a estimativa do deslocamento angular pode ser obtida fazendo o integral de Ω, num dado intervalo de tempo, segundo um eixo:

$$\varphi_Z = \int_{t_0}^{t_1} \Omega_Z \, dt + \varphi_{Z0} = \Omega_Z \cdot (t_1 - t_0) + \varphi_{Z0}$$

Figura 9 - formula usada para a implementação do deslocamento angular

Display I2C

O display I2C é um de display de 2 linhas com 16 caracteres em cada linha. O display disponibiliza uma interface I2C para comunicação com um microcontrolador. Por cada byte enviado via I2C, a interface I2C-paralelo define 4 sinais de dados para o display e 4 sinais de controlo de acordo com a figura 5. O endereço I2C do display é o 27h.

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

Figura 10 endereço dos caracteres do display(16x2)

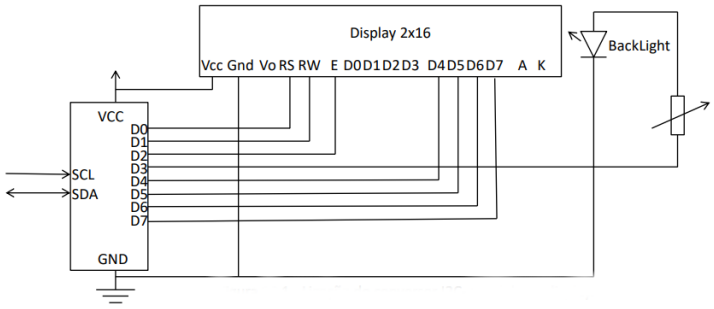

Figura 11 - Ligação do conversor I2C

Figura 12 - Display I2C

Instruções do Display

Encontra-se aqui a tabela usada com os respetivos endereços para as instruções usadas, para a construção do programa em Arduino para o display.

Função	Tipo	D7	D6	D5	D4	D3	D2	D1	D0	Tempo
Clear Display	Comando	0	0	0	0	0	0	0	1	5ms
Cursor Home	Comando	0	0	0	0	0	0	0	0	5ms
Entry Mode	Comando	0	0	0	0	0	1	I/D	S	5us
Display On/Off	Comando	0	0	0	0	1	D	C	B	120us
Cursor & Display shift	Comando	0	0	0	1	S/C	R/L	x	x	120us
Function set	Comando	0	0	1	DL	N	F	x	x	120us
Set Cursor Adress	Comando	1	A6	A5	A4	A3	A2	A1	A0	120us
Write Data to cursor Position	Dados	D7	D6	D5	D4	D3	D2	D1	D0	120us

Tabela 2 - Instruções do Display

- **Legenda:**

I/D = 1 - incrementa a posição do cursor.

I/D= 0 - decrementa a posição do cursor.

S= 1 - permite display shift.

D= display, C= cursor, B= blink (1= ON, 0= Off).

S/C= 1 - display shift, S/C= 0 - cursor move.

R/L= 1 - shift right, R/L= 0 - shift left.

DL= 1 - 8 bits de data, DL= 0 - 4 bits de data.

N= 1 - display 2 linhas, N= 0 - display de 1 linha.

F=1 - (5x10 dots só em 1 linha), F= 0 (5x7 dots em 1 ou 2 linhas).

Diagrama de ligações

Antes de passar para a implementação no Arduino do sensor e do display, o grupo optou por fazer um diagrama de ligações entre o Arduino, o sensor e o display, para ter uma melhor perceção do tipo de ligações que irão ser feitas e quais os pinos associados ao display e o sensor. A única diferença na ligação entre os dois dispositivos é que o sensor tem de estar ligado aos 3.3V e o display aos 5V. Os pinos A4 e A5 são comuns a ambos, pois estamos a trabalhar com o protocolo I2C, e o GND.

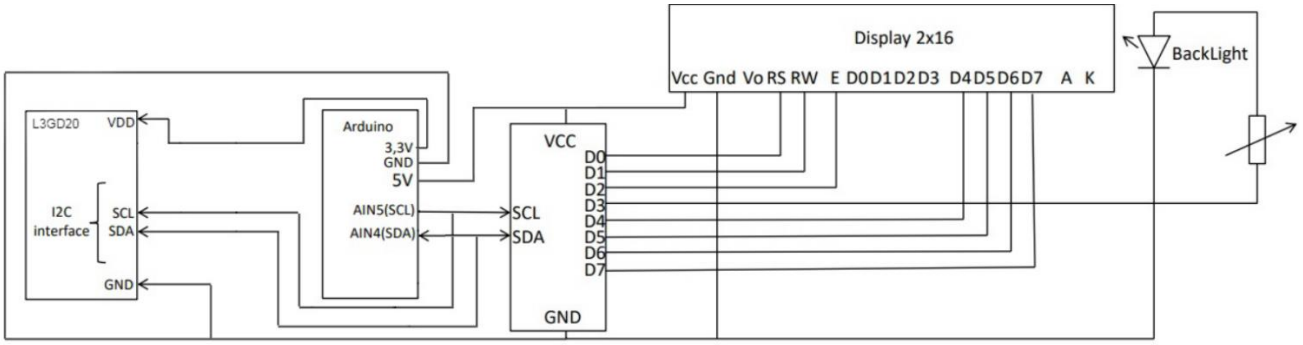

Figura 13 - diagrama de ligações entre o sensor L3GD20, Arduino e display

Programa no Arduino

Para representar a informação do sensor no display, e ler a informação do sensor, foram concebidos dois programas para ambos os dispositivos. A informação para a representação do sensor no display foi feita referente à informação da tabela acima e das folhas do Engenheiro Jorge Pais. A consulta dos métodos foi feita referente às folhas do Engenheiro Carlos Carvalho.

Em cada um dos métodos abaixo, as respetivas funções de cada método estavam codificados a 8 ou a 4 bits, cada instrução tinha de ser convertida de binário para hexadecimal, pelo que foi usado o website www.calculadoraonline.com para essa respetiva conversão.

O método *escreverDados4(byte quatroBits)* serve para enviar caracteres para o display. Recorrendo à biblioteca *Wire* e às várias funções dessa biblioteca, é sempre escrito os valores endereços das variáveis. Usa-se a função *Wire.beginTransmission()* para começar a transmissão para o dispositivo I2C slave, passando como parâmetro o valor do endereço do display. De seguida é usada a função *Wire.write* que escrever o valor do endereço, neste caso irá ser realizada uma operação XOR bit a bit entre as variáveis, e de seguida usando a função *Wire.endTransmission()* para terminar a transmissão de

dados. É de notar que é necessário fazer este processo três vezes para ler os diferentes endereços do display.

O método *escreverDados8(byte oitoBits)* serve para enviar um caracter para o display, para isso recorreu-se à função *escrecerDados4* e recebendo como argumento o *byte oitoBits*, fazendo um shift right de 4, ou seja, recorreu-se a máscaras de bits para "shiftar" os bits de modo a enviar o primeiro bit de maior peso. E como estamos a falar de uma função que escreve a 8 bits, teve-se que chamar mais uma vez o método *escreverDados4* para escrever o valor dos oitoBits.

```
void escreverDados8(byte oitoBits) {
  escreverDados4(oitoBits >> 4);
  escreverDados4(oitoBits & 0xF);
}
```

O método *escreverComandos4(byte quatroBits)* serve para enviar comandos para o display. Recorrendo à biblioteca *Wire* e às várias funções dessa biblioteca, é sempre escrito os valores endereços das variáveis. Usa-se a função *Wire.beginTransmission()* para começar a transmissão para o dispositivo I2C slave, passando como parâmetro o valor do endereço do display. De seguida é usada a função *Wire.write()* que escrever o valor do endereço, neste caso irá ser realizada uma operação XOR bit a bit entre as variáveis, e de seguida usando a função *Wire.endTransmission()* para terminar a transmissão de dados. É de notar que é necessário fazer este processo três vezes para ler os diferentes endereços do display.

O método *escreverComandos8(byte oitoBits* serve para enviar um comando para o display, para isso recorreu-se à função *escrecerDados4* e recebendo como argumento o *byte oitoBits*, fazendo um shift right de 4, ou seja, recorreu-se a máscaras de bits para "shiftar" os bits de modo a enviar o primeiro bit de maior peso. E como estamos a falar de uma função que escreve a 8 bits, teve-se que chamar mais uma vez o método *escreverDados4* para escrever o valor dos oitoBits.

```
void escreverComandos8(byte oitoBits) {
  escreverComandos4(oitoBits >> 4);
  escreverComandos4(oitoBits & 0xF);
}
```

O método *displayInit()* servirá para inicializar o display para funcionar em modo 4bits. Como primeiro passo para a construção deste método, recorreu-se ao método *Wire.begin()* inicializando a biblioteca e juntar o bus I2C como master ou slave. De seguida foi usado as funções *escreverComando4()* e *escreverComandos8()* para ler e escrever comandos com a informação acerca dos endereços, *delay()* e *delayMicrosends()* fazendo uma espera quando ler a próxima instrução do respetivo endereço.

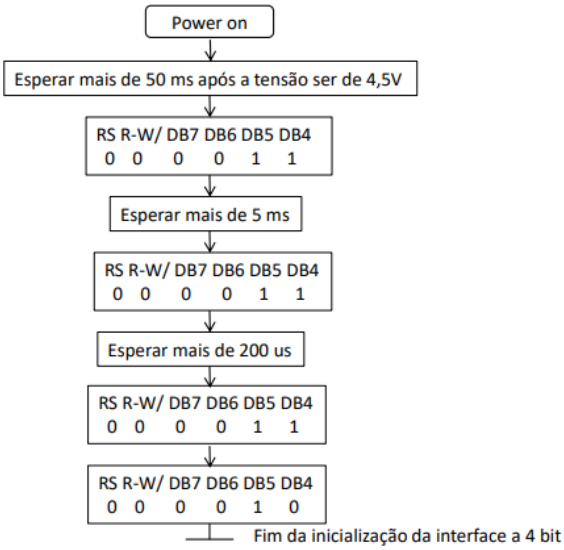

Figura 14 diagrama de ligações acerca dos diferentes comandos e dados a serem lidos, referente à construção do método displayInit()

```
void displayInit() {
  Wire.begin();
  delay(50);
  escreverComandos4(0x03);
  delay(5);
  escreverComandos4(0x03);
  delayMicroseconds(200);

  escreverComandos8(0x32);

  //Funcao set
  escreverComandos4(0x02);
  escreverComandos4(0x08);

  delayMicroseconds(120);

  displayCursorOffBlinckOff();

  displayCursorOnBlinckOn();

  //set cursor
  escreverComandos4(0x08);
  escreverComandos4(0x00);

  delayMicroseconds(120);

}
```

O método *displayClear()* que irá servir para limpar o display. Foi usado a função *escreverComandos8()* para ler o respetivo endereço.

```
void displayClear() {
  escreverComandos8(0x01);
  delay(5);
}
```

O método *displaySetCursor(byte linha, byte coluna)* serve para colocar a informação pretendida na certa linha e coluna. Para isso foi criado uma condição *if* para ler o número de linhas e colunas. A seguir foram criadas duas variáveis do tipo byte: *MSB e LSB.* Estas variáveis iram guardar o valor da linha e coluna, para a linha foi necessário fazer uma operação XOR bit a bit, entre o número de linhas e o endereço 0x8, para conseguir todos os endereços das respetivas linhas. No final recorreu-se às funções *escreverComandos4()* para ler a respetiva informação acerca das variáveis *MSB e LSB.*

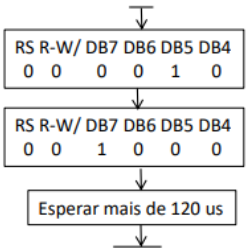

Figura 15 - diagrama de ligações acerca dos diferentes comandos e dados a serem lidos, referente à construção do método displaySetCursor()

```
void displaySetCursor(byte linha, byte coluna) {
  if ( 0 <= linha <= 2 and  0 <= coluna <= 16) {
    byte MSB = (byte)((linha * 4) | 0x8);
    byte LSB = (byte) coluna;
    escreverComandos4(MSB);
    escreverComandos4(LSB);
    delayMicroseconds(120);
  }
}
```

Os métodos *displayCursorOffBlinckOff() e displayCursorOnBlinckOn()* servem para colocar o curso no display e permitir fazer o efeito de blinck, aparece e desaprece consoante um certo tempo. Para isso foi usada a função *escreverComandos8()* para enviar os respetivos endereços acerca destes respetivos métodos, e aplicando um delay em ambos os métodos para aparecer e desaparecer ao fim de um certo tempo.

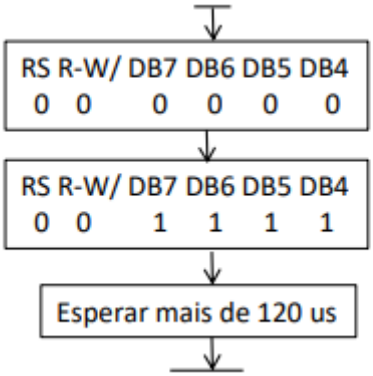

Figura 16 - diagrama de ligações acerca dos diferentes comandos e dados a serem lidos, referente à construção do método displayCursorOffBlinckOff() e *displayCursor*OnBlinck*On()*

Nota: para a construção da função *displayCursorOffBlinckOff()* recorreu-se à tabela acima, o processo usado para obter o respetivo endereço foi simplesmente colocar os bits D2,D1 e D0 a 0.

```
void displayCursorOffBlinckOff() {
  escreverComandos8(0x08);
  delayMicroseconds(120);
}

void displayCursorOnBlinckOn() {
  escreverComandos8(0x0F);
  delayMicroseconds(120);
}
```

O método *displayPrintChar(char c)* que permite imprimir um caracter através de um certo endereço. Foi usado a função *escreverComandos8()* para enviar e ler o comando o char c , sendo esse char um respetivo endereço, que irá ser passado como argumento da função.

```
void displayPrintChar(char c) {
  escreverDados8(c);
  delayMicroseconds(120);
}
```

Por último o método *displayPrintString(String s)* que permite imprimir uma String no display. Para isso usou – se um ciclo *for* no qual lê o comprimento da string, passada como argumento, e no final recorrendo à função *displayPrintChar()* para ler o comprimento e os valores passados no ciclo *for*.

```
void displayPrintString(String s) {
  for (int i = 0; i < s.length(); i++) {
    displayPrintChar(s[i]);
  }
}
```

Com o programa para representar as informações no display feitas, passou-se para a construção de outro programa capaz de ler as informações e dos respetivos valores.

O método *byte L3GD20_read_8bit_value(byte regAddress)* serve para ler um valor a 8bit no sensor. Para isso recorreu-se mais uma vez à biblioteca *Wire* para permitir a ligação do protocolo I2C. Como primeiro passo usou-se a função *Wire.beginTransmission()* para começar a transmissão, e passando como parâmetro o endereço do sensor em estudo. De seguida usa-se a função *Wire.write()* para escrever o valor do endereço, neste caso irá ser o valor que é passado como argumento da função. De seguida termina-se a transmissão com o comando *Wire.endTransmission()*. De seguida usou-se a função *Wire.requestFrom()* no qual, permite pedir ao endereço do

nosso sensor, pedir uma determinada quantidade de bytes, neste caso foi pedido 1byte. No final, usou-se a função *Wire.available()*, permitindo retornar uma determinada quantidade de bytes. Neste caso optou-se por usar um ciclo enquando *Wire.available* for menor que zero. Caso seja verdade, é retornada do final da função a leitura dos respetivos bytes, através do uso da função *Wire.read()*.

```
byte L3GD20_read_8bit_value(byte regAddress) {

  Wire.beginTransmission(L3GD20_ENDERECO);
  Wire.write(regAddress);
  Wire.endTransmission();

  Wire.requestFrom(L3GD20_ENDERECO, 1);
  while (Wire.available() < 0)
    return Wire.read();

}
```

O método *void L3GD20_write_8bit_value(byte regAddress,byte value)* serve para escrever valores a 8bits. A implementação deste método é simples, não tão complexa como a anterior.Nesta função começou-se por ler o valor do endereço do sensor, usando a função *Wire.beginTransmission()* e de seguida escrever os seus respetivos endereços, usando a biblioteca *Wire.write()*, no final terminar a transmissão, usando o comando *Wire.endTransmission()*.

```
void L3GD20_write_8bit_value(byte regAddress, byte value) {

  Wire.beginTransmission(L3GD20_ENDERECO);
  Wire.write(regAddress);
  Wire.write(value);
  Wire.endTransmission();

}
```

O método *bool L3GD20_Init()* serve para inicializar o display, como primeiro passou chamou-se a função *L3GD20_write_8bit_value()* permitindo escrever o valor do sensor com o endereço 0x0F. De seguida criou-se uma condição *if*, caso o valor do sensor for igual ao valor da constante *WHO_AM_I* retorna *true*, caso contrário retorna false.

```
bool L3GD20_Init() {


  L3GD20_write_8bit_value(sensorInit, 0x0F);

  if (L3GD20_read_8bit_value(0x0F) == WHO_AM_I) {
    return true;
  } else {
    return false;
  }
}
```

O método *int L3GD20_X_channel_Read()* serve para ler o valor do eixo X do giroscópio. Para a implementação desta função, foi primeiro saber-se qual o endereço do eixo X, sendo que este eixo continha dois endereços, servindo para ler o valor mais alto e outro mais baixo. Por isso foram criadas duas variáveis do tipo byte para guardar os valores do xLow e xHigh. Para ler cada endereço, foi usada a função *L3GD20_read_8bit_value()*. No final foi criada uma variável do tipo *int* para realizar uma operação XOR bit a bit entre o valor do endereço de xLow e o valor do endereço de xHigh, multiplicado também pelo fator de sensibilidade. E retornada o valor dessa variável do tipo int.

```
int L3GD20_X_channel_Read() {

  byte xLow = L3GD20_read_8bit_value(OUT_X_LOW);

  byte xHigh = L3GD20_read_8bit_value(OUT_X_HIGH);

  float fs = fatorSensitividade(250);

  int velX = (int) (xHigh << 8 | xLow) * fs;

  return velX;

}
```

O método *int L3GD20_Y_channel_Read()* serve para ler o valor do eixo Y do giroscópio. Para a implementação desta função, foi primeiro saber-se qual o endereço do eixo Y, sendo que este eixo continha dois endereços, servindo para ler o valor mais alto e outro mais baixo. Por isso foram criadas duas variáveis do tipo byte para guardar os valores do yLow e yHigh. Para ler cada endereço, foi usada a função *L3GD20_read_8bit_value()*. No final foi criada uma variável do tipo *int* para realizar uma operação XOR bit a bit entre o valor do endereço de xLow e o valor do endereço de yHigh multiplicado também pelo fator de sensibilidade. E retornada o valor dessa variável do tipo int.

```
int L3GD20_Y_channel_Read() {
  byte yLow = L3GD20_read_8bit_value(OUT_Y_LOW);
  byte yHigh = L3GD20_read_8bit_value(OUT_Y_HIGH);

  float fs = fatorSensitividade(250);

  int velY = (int) (yHigh << 8 | yLow) * fs;

  return velY;
}
```

O método *int L3GD20_Z_channel_Read()* serve para ler o valor do eixo Z do giroscópio. Para a implementação desta função, foi primeiro saber-se qual o endereço do eixo Y, sendo que este eixo continha dois endereços, servindo para ler o valor mais alto e outro mais baixo. Por isso foram criadas duas variáveis do tipo byte para guardar os valores do zLow e zHigh. Para ler cada endereço, foi usada a função *L3GD20_read_8bit_value()*. No final foi criada uma variável do tipo *int* para realizar uma operação XOR bit a bit entre o valor do endereço de zLow e o valor do endereço de zHigh multiplicado também pelo fator de sensibilidade. E retornada o valor dessa variável do tipo int.

```
int L3GD20_Z_channel_Read() {

  byte zLow = L3GD20_read_8bit_value(OUT_Z_LOW);

  byte zHigh = L3GD20_read_8bit_value(OUT_Z_HIGH);

  float fs = fatorSensitividade(250);

  int velZ = (int) (zHigh << 8 | zLow) * fs;

  return velZ;
}
```

O método *byte L3GD20_Temperature_Read()* permite retornar o valor da temperatura. Para isso usou-se a função *L3GD20_read_8_bit_valur()* para ler o respetivo endereço da temperatura.

```
byte L3GD20_Temperature_Read() {
  return 28 - L3GD20_read_8bit_value(OUT_TEMP);
}
```

Por último a função *float fatorSensitividade(int v)* que serve para adquirir os valores acerca do fator de sensibilidade. Existem 3 valores para serem escolhidos, que são : 250, 500 e 2000. Caso seja escolhido um destes valores, e depois usado o valor associado, que vai ser multiplicado pelos três eixos.

```
float fatorSensitividade(int v) {
  float valor = 0.0;

  if (v = 250) {
    valor = 8.75 * pow(10, -3);
  } else if (v == 500) {
    valor = 17.50 * pow(10, -3);
  } else if (v == 2000) {
    valor = 70 * pow(10, -3);
  }
  return valor;
```

Diagrama de Atividades - Swimlanes

As swimlanes servem para fazer a partição dos diagramas de atividade da forma mais adequada. Esta partição definirá as unidades funcionais, designadas como processos que serão implementados com autómatos. Neste caso temos 5 swimlanes, *Sistema*, *Temperatura*, *AngZ*, *AngY* e *AngX*. Cada swimlane é composta por um autómato com 2 ou mais estados e todos os autómatos estão a correr em simultâneo. Como podemos observar na figura, o autómato inicial é o Sistema em que o estado que começa é o iniciar. O estado final desta atividade é o Finalizar que se encontra também dentro do autómato Sistema.

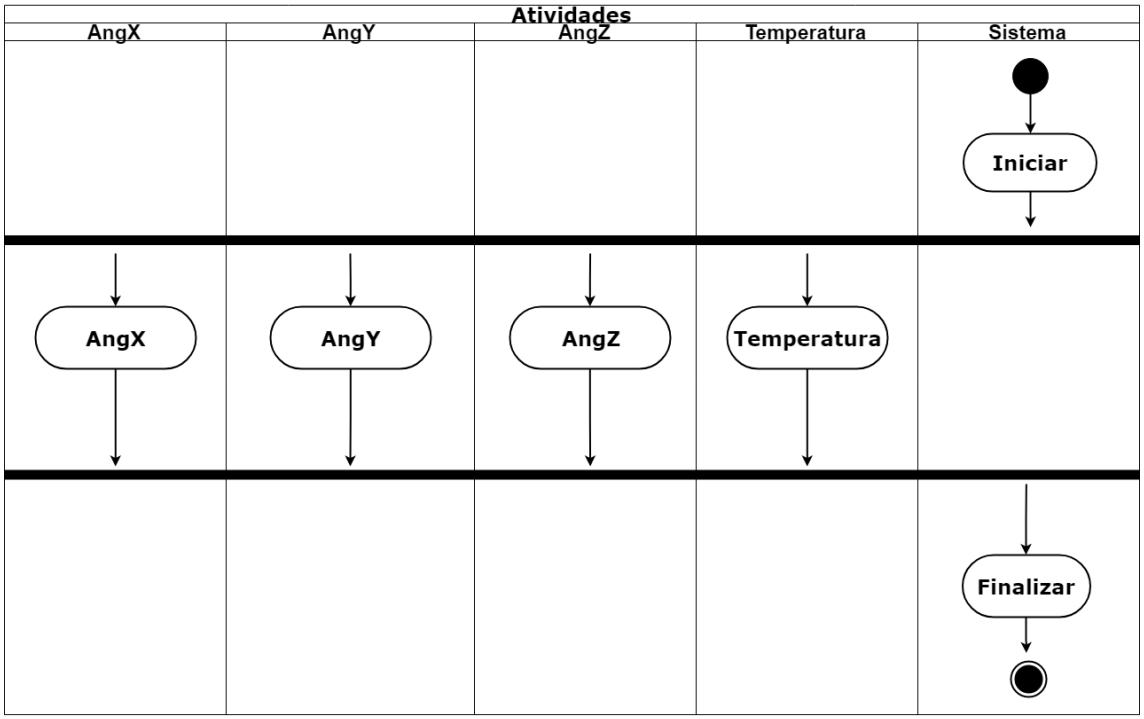

Figura 17 - Diagrama de Atividades - Swimlanes

Diagrama de Atividades – Fluxo de Objetos

As atividades podem receber e enviar objetos e até podem modificar o estado dos objetos. Esta transferência de objetos entre as atividades são representadas no seguinte diagrama de atividade. Neste caso vamos ter 5 objetos cujo valor booleano vai ser alterado, *haPedido*, *angvalX*, *angvalY*, *angvalZ*, *tempVal*.

O valor booleano de *haPedido* é alterado quando passam 100 milissegundos desde o estado iniciar do autómato Sistema e faz com que nos outros autómatos o estado vá para *Join*. Quando o valor dos três últimos objetos fica a False, o estado *Finalizar* muda a condição.

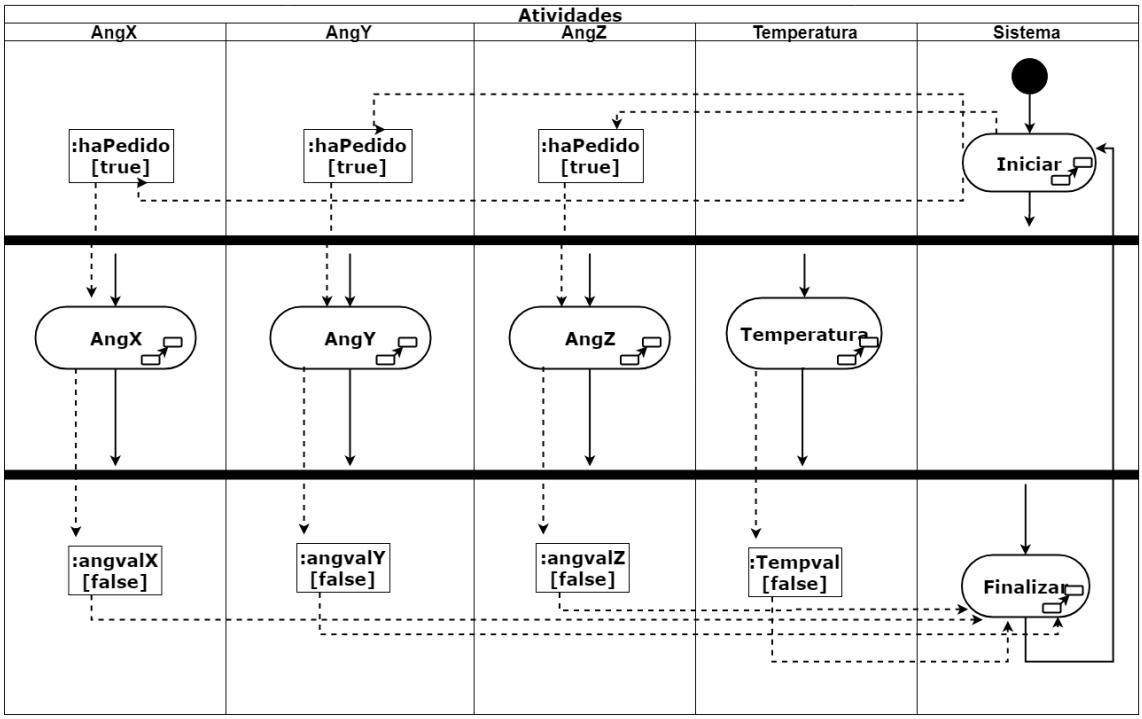

Figura 18 - Diagrama de Atividades - Fluxo de Objetos

Diagrama de Atividades – Explicação

O diagrama de atividades tem 5 autómatos, que são todos executados ao mesmo tempo, e o Sistema entra no estado *Iniciar* e os restantes nos respetivos Fork's. Quando passam 100 milissegundos o autómato Sistema passa para o estado espera e altera o valor do *haPedido*. Os outros autómatos começam por meter o valor dos seus respetivos objetos em *true* e quando *haPedido* entra no estado buscar que chama as funções de obter os valores da velocidade angular e calcula o valor do ângulo. Depois entram no estado *join* que altera de novo o valor do objeto para false. Quando todos os objetos forem false o autómato Sistema entra no estado *Finalizar* e realiza as funções *escreverDisplay* e *escreverPython*, passando como parâmetro os valores obtidos nos estados Buscar.

Programa em Python

Elaborou-se um programa em *python* para a representação de valores obtidos pelo Arduino. Fez-se os import necessários ao funcionamento do programa. A biblioteca "serial" permite ler os dados do Arduino , a "time"e "datetime" permitem fazer medições temporais , o "pygame" serve para tratar da interface que representa os dados, e "sys" que permite fechar o programa assim que terminado.

Inicializa-se as variáveis a utilizar , como os "timers" , as cores a serem utilizadas na interface , e o carregamento de uma imagem de fundo , definiu-se a porta do Arduino (neste caso porta COM5) e a sua velocidade de transmissão de dados (em baudrate com o valor de 9600 neste caso), por ultimo define-se a função "Window" que cria a janela da interface (figura 13).

```
import serial
import time
import pygame
import sys
import datetime
pygame.init()


screen = pygame.display.set_mode([800,800])
timer = pygame.time.Clock()
t0 = pygame.time.get_ticks()


Black =(0,0,0)
cor_fundo = (255,255,255)
fundo = pygame.image.load("fundo.jpg")


def Window():
    screen.fill(cor_fundo)
    screen.blit(fundo, [0,0])


com = 'COM5'
baudrate = 9600
```

De seguida usando a função “comInit” estabeleceu-se a ligação ao Arduino, imprimindo na consola uma mensagem em caso de sucesso e outra em caso de insucesso. A função “StringReceive” devolve os dados do Arduino ou dispara um print na consola em caso de erro.

```
def comInit(com, baudrate):
    try:
        Serie = serial.Serial(com, baudrate)
        print ("Sucesso na ligacao ao Arduino")
        print ("Ligado ao " + Serie.portstr)
        return Serie
    except Exception as e:
        print ("Insucesso na ligacao ao Arduino")
        print (e)
        return None


def stringReceive(Serie):
    try:
        return Serie.readline().strip()
    except Exception as e:
        print("Erro na comunicacao (stringReceive)")
        print (e)
        Serie.close()
```

As funções “temp”, “tempTexto”, “press”, “pressTexto” e “tick” são usadas para definir a fonte, tamanho e posição das letras a serem escritas na janela. Inicializam-se mais variáveis a serem utilizadas a seguir e chama-se a função “Window” para gerar a janela.

```
def temp(txt, size):
    Fonte = pygame.font.Font("times.ttf", size)
    Texto = Fonte.render(txt, True, Black)
    screen.blit(Texto, [7, 160])
```

```
def tempTexto(txt, size):
    Fonte = pygame.font.Font("times.ttf", size)
    Texto = Fonte.render(txt, True, Black)
    screen.blit(Texto, [110, 230])

def press(txt, size):
    Fonte = pygame.font.Font("times.ttf", size)
    Texto = Fonte.render(txt, True, Black)
    screen.blit(Texto, [10, 325])

def pressTexto(txt, size):
    Fonte = pygame.font.Font("times.ttf", size)
    Texto = Fonte.render(txt, True, Black)
    screen.blit(Texto, [110, 395])

def tick(txt,size):
    Fonte = pygame.font.Font("DS-DIGI.ttf",size)
    Texto = Fonte.render(txt, True, Black)
    screen.blit(Texto, [400, 400])

s = comInit(com, baudrate)
print (s)
a = [None]*5
imprimir_1_vez = True
imprimir_cada_30sec = True

Window()
```

Através do ciclo “while” elaborado, vai-se marcando o tempo depois de recebidos os valores do Arduino. Caso a variável “imprimir_1_vez” tomar o valor True , imprime os valores lidos do Arduino usando os métodos de escrita anteriores uma vez.

```
while (s != None):
    now = datetime.datetime.now()
    if( now.hour < 10):
        b = ("0" + str(now.hour))
```

```
    else:
        b = str(now.hour)
    if( now.minute < 10):
        c = ("0" + str(now.minute))


    else:
        c = str (now.minute)
    if( now.second < 10):
        d = ("0" + str(now.second))
    else:
        d = str (now.second)
    tick( b + ":" + c + ":" + d,70)
    for indice in range(5):
        a[indice] = stringReceive(s)
    if (imprimir_1_vez == True):
        Window();
        temp(a[0], 55)
        print(a[0])
        tempTexto(a[1], 70)
        print(a[1])
        press(a[2] , 55)
        print(a[2])
        pressTexto(a[3], 70)
        print(a[3])
        imprimir_1_vez = False
```

Finalmente ao fim de um certo tempo, os dados são atualizados, ou seja são impressos a cada 30 segundos e ainda desenha uma linha de acordo com a temperatura lida (estilo termómetro). Define-se um evento que termina o programa assim que é fechada a janela.

```
t1 = pygame.time.get_ticks()
    if (t1 - t0 > 30000):
        if (imprimir_cada_30sec == True):
            Window();
            temp(a[0], 55)
```

```
            print(a[0])
            pygame.draw.line(screen,(255,0,0),(30,30),(15+a[0],30),width=6)
            tempTexto(a[1], 70)
            print(a[1])
            press(a[2] , 55)
            print(a[2])
            pressTexto(a[3], 70)
            print(a[3])
            a[4] = None
            imprimir_cada_30sec = False
        for c in a:
            if c == None:
                a = [None] * 5
                t0 = t1
                imprimir_cada_30sec = True
    for event in pygame.event.get():
       if  event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()


    pygame.display.update()
    timer.tick(60)
pygame.quit()
```

Código Arduino

AUTÓMATOS

```
//SISTEMA
int estadoSistema;
const int iniciar = 0;
const int espera = 1;
const int finalizar = 2;
boolean haPedido;

//TEMPERATURA
int estadoTemp;
const int ForkTemp = 0;
const int BuscarT = 1;
const int JoinTemp = 2;

byte valorTemp;
boolean tempval;

//ANGULAR_X
int estadoAngX;
const int ForkAngX = 0;
const int BuscarX = 1;
const int JoinAngX = 2;

boolean angvalX;
int valorX;
int phiX;

//ANGULAR_Y
int estadoAngY;
const int ForkAngY = 0;
const int BuscarY = 1;
const int JoinAngY = 2;

boolean angvalY;
int valorY;
int phiY;

//ANGULAR_Z
int estadoAngZ;
const int ForkAngZ = 0;
const int BuscarZ = 1;
const int JoinAngZ = 2;

boolean angvalZ;
int valorZ;
int phiZ;

unsigned long t0, t1x, t1y, t1z;
```

```
void Sistema() {

  switch (estadoSistema) {
    case iniciar:
      AtividadeIniciar();
      estadoSistema = espera;
      t0 = millis();

      break;

    case espera :
      if (millis() - t0 >= 100){

        haPedido = true;
        estadoSistema = finalizar;

      }

      break;

    case finalizar:

      if (!angvalX && !angvalY && !angvalZ && !tempval) {

        escreverDisplay(valorX, valorY, valorZ, valorTemp);
        escreverPython(valorTemp, phiX, phiY, phiZ);
        estadoSistema = iniciar;
        haPedido = false;

      }

      break;
  }

}

void AtividadeIniciar() {

  estadoTemp = ForkTemp;
  estadoAngX = ForkAngX;
  estadoAngY = ForkAngY;
  estadoAngZ = ForkAngZ;
}

void Temp() {

  switch (estadoTemp) {

    case ForkTemp:

      tempval = true;
      estadoTemp = BuscarT;

      break;

    case BuscarT:

      valorTemp = L3GD20_Temperature_Read();
      estadoTemp = JoinTemp;

      break;

    case JoinTemp:
      tempval = false;
```

```
      break;

  }
}

void AngX() {

  switch (estadoAngX) {

    case ForkAngX:
      angvalX = true;
      if (haPedido)
        estadoAngX = BuscarX;

      break;

    case BuscarX:
      valorX = L3GD20_X_channel_Read();
      t1x = millis();
      phiX = valorX * (t1x - t0) + phiX;

      estadoAngX = JoinAngX;
      break;

    case JoinAngX:
      angvalX = false;
      break;

  }
}

void AngY() {

  switch (estadoAngY) {

    case ForkAngY:
      angvalY = true;
      if (haPedido)
        estadoAngY = BuscarY;

      break;


    case BuscarY:
      valorY = L3GD20_Y_channel_Read();
      phiY = valorY * (t1y - t0) + phiY;
      estadoAngY = JoinAngY;
      break;

    case JoinAngY:
      angvalY = false;
      break;

  }
}
```

```
void AngZ() {

  switch (estadoAngZ) {

    case ForkAngZ:
      angvalZ = true;
      if (haPedido)
        estadoAngZ = BuscarZ;
      break;

    case BuscarZ:
      valorZ = L3GD20_Z_channel_Read();
      phiZ = valorZ * (t1z + t0) + phiZ;
      estadoAngZ = JoinAngZ;
      break;

    case JoinAngZ:
      angvalZ = false;
      break;

  }
}

void escreverDisplay(int X, int Y, int Z, byte T) {

  displayInit();
  displayClear();
  displaySetCursor(0, 0);
  displayPrintChar(0xF4);
  displayPrintString("x:" + String(X));

  displayPrintChar(0xDF);
  displayPrintChar(0x2F);
  displayPrintChar(0x73);

  displayPrintString(" ");
  displayPrintChar(0xF4);
  displayPrintString("y:" + String(Y));

  displayPrintChar(0xDF);
  displayPrintChar(0x2F);
  displayPrintChar(0x73);

  displaySetCursor(1, 0);
  displayPrintChar(0xF4);
  displayPrintString("z:" + String(Z));

  displayPrintChar(0xDF);
  displayPrintChar(0x2F);
  displayPrintChar(0x73);

  displayPrintString(" T:" + String(T));
  displayPrintChar(0xDF);
  displayPrintChar(0x43);

}

void escreverPython(byte Temp, int PX, int PY, int PZ){
  Serial.println("PhiX: " + String(PX) + "graus" );
  Serial.println("PhiY: " + String(PY) + "graus" );
  Serial.println("PhiZ: " + String(PZ) + "grau" );
  Serial.println("Temp: " + String(Temp) + "grau centigrado" );
}
```

```
void setup() {
  Wire.begin();
  Serial.begin(9600);
  angvalX = false;
  angvalY = false;
  angvalZ = false;
  tempval = false;
  estadoSistema = iniciar;
  estadoTemp = ForkTemp;
  estadoAngX = ForkAngX;
  estadoAngY = ForkAngY;
  estadoAngZ = ForkAngZ;

  }

  void loop() {
//  displayInit();
//  displayClear();
  Temp();
  AngX();
  AngY();
  AngZ();
  Sistema();


  }
```

SENSOR

```
#include <Wire.h>

#define L3GD20_ENDERECO 0x69

#define OUT_X_LOW 0x28
#define OUT_X_HIGH 0x29

#define OUT_Y_LOW 0x2A
#define OUT_Y_HIGH 0x2B

#define OUT_Z_LOW 0x2C
#define OUT_Z_HIGH 0x2D

#define OUT_TEMP 0x26

#define sensorInit 0x20

#define WHO_AM_I 0x0F

byte L3GD20_read_8bit_value(byte regAddress) {

  Wire.beginTransmission(L3GD20_ENDERECO);
  Wire.write(regAddress);
  Wire.endTransmission();

  Wire.requestFrom(L3GD20_ENDERECO, 1);
  while (Wire.available() < 0)
    return Wire.read();

}


void L3GD20_write_8bit_value(byte regAddress, byte value) {

  Wire.beginTransmission(L3GD20_ENDERECO);
  Wire.write(regAddress);
  Wire.write(value);
  Wire.endTransmission();

}

bool L3GD20_Init() {


  L3GD20_write_8bit_value(sensorInit, 0x0F);

  if (L3GD20_read_8bit_value(0x0F) == WHO_AM_I) {
    return true;
  } else {
    return false;
  }
}
```

```
int L3GD20_X_channel_Read() {

  byte xLow = L3GD20_read_8bit_value(OUT_X_LOW);

  byte xHigh = L3GD20_read_8bit_value(OUT_X_HIGH);

  float fs = fatorSensitividade(250);

  int velX = (int) (xHigh << 8 | xLow) * fs;

  return velX;

}


int L3GD20_Y_channel_Read() {
  byte yLow = L3GD20_read_8bit_value(OUT_Y_LOW);
  byte yHigh = L3GD20_read_8bit_value(OUT_Y_HIGH);

  float fs = fatorSensitividade(250);

  int velY = (int) (yHigh << 8 | yLow) * fs;

  return velY;
}


int L3GD20_Z_channel_Read() {

  byte zLow = L3GD20_read_8bit_value(OUT_Z_LOW);

  byte zHigh = L3GD20_read_8bit_value(OUT_Z_HIGH);

  float fs = fatorSensitividade(250);

  int velZ = (int) (zHigh << 8 | zLow) * fs;

  return velZ;
}



byte L3GD20_Temperature_Read() {
  return 28 - L3GD20_read_8bit_value(OUT_TEMP);
}

float fatorSensitividade(int v) {
  float valor = 0.0;

  if (v = 250) {
    valor = 8.75 * pow(10, -3);
  } else if (v == 500) {
    valor = 17.50 * pow(10, -3);
  } else if (v == 2000) {
    valor = 70 * pow(10, -3);
  }
  return valor;
}
```

DISPLAY

```
#define RS 0x01
#define RW 0x02
#define EN 0x04
#define LUZ 0x08
#define ENDERECO 0x27
#include <Wire.h>

void escreverDados4(byte quatrobits) {
  Wire.beginTransmission(ENDERECO);
  Wire.write((quatrobits << 4) | LUZ | RS);
  Wire.endTransmission();
  Wire.beginTransmission(ENDERECO);
  Wire.write((quatrobits << 4) | LUZ | RS | EN );
  Wire.endTransmission();
  delayMicroseconds(1);
  Wire.beginTransmission(ENDERECO);
  Wire.write((quatrobits << 4) | LUZ | RS);
  Wire.endTransmission();
  delayMicroseconds(40);
}

void escreverComandos4(byte quatrobits) {
  Wire.beginTransmission(ENDERECO);
  Wire.write((quatrobits << 4) | LUZ);
  Wire.endTransmission();
  Wire.beginTransmission(ENDERECO);
  Wire.write((quatrobits << 4) | LUZ | EN );
  Wire.endTransmission();
  delayMicroseconds(1); // enable ativo >450ns
  Wire.beginTransmission(ENDERECO);
  Wire.write((quatrobits << 4) | LUZ );
  Wire.endTransmission();
  delayMicroseconds(40); // tempo > 37us para comando fazer efeito
}


void escreverDados8(byte oitoBits) {
  escreverDados4(oitoBits >> 4);
  escreverDados4(oitoBits & 0xF);
}

void escreverComandos8(byte oitoBits) {
  escreverComandos4(oitoBits >> 4);
  escreverComandos4(oitoBits & 0xF);
}

void displayInit() {
  Wire.begin();
  delay(50);
  escreverComandos4(0x03);
  delay(5);
  escreverComandos4(0x03);
  delayMicroseconds(200);
```

```
  escreverComandos8(0x32);

  //Funcao set
  escreverComandos4(0x02);
  escreverComandos4(0x08);

  delayMicroseconds(120);

  displayCursorOffBlinckOff();

  displayCursorOnBlinckOn();

  //set cursor
  escreverComandos4(0x08);
  escreverComandos4(0x00);

  delayMicroseconds(120);

}

void displayClear() {
  escreverComandos8(0x01);
  delay(5);
}

void displaySetCursor(byte linha, byte coluna) {
  if ( 0 <= linha <= 2 and  0 <= coluna <= 16) {
    byte MSB = (byte)((linha * 4) | 0x8);
    byte LSB = (byte) coluna;
    escreverComandos4(MSB);
    escreverComandos4(LSB);
    delayMicroseconds(120);
  }
}

void displayCursorOffBlinckOff() {
  escreverComandos8(0x08);
  delayMicroseconds(120);
}

void displayCursorOnBlinckOn() {
  escreverComandos8(0x0F);
  delayMicroseconds(120);
}

void displayPrintChar(char c) {
  escreverDados8(c);
  delayMicroseconds(120);
}



void displayPrintString(String s) {
  for (int i = 0; i < s.length(); i++) {
    displayPrintChar(s[i]);
  }
}
```

Código Python

```
import serial
import time
import pygame
import sys
import datetime
pygame.init()

screen = pygame.display.set_mode([800,800])
timer = pygame.time.Clock()
t0 = pygame.time.get_ticks()

Black =(0,0,0)
cor_fundo = (255,255,255)
fundo = pygame.image.load("fundo.jpg")

def Window():
    screen.fill(cor_fundo)
    screen.blit(fundo, [0,0])

com = 'COM5'
baudrate = 9600
def comInit(com, baudrate):
    try:
        Serie = serial.Serial(com, baudrate)
        print ("Sucesso na ligacao ao Arduino")
        print ("Ligado ao " + Serie.portstr)
        return Serie
    except Exception as e:
        print ("Insucesso na ligacao ao Arduino")
        print (e)
        return None
```

```
def stringReceive(Serie):
    try:
        return Serie.readline().strip()
    except Exception as e:
        print("Erro na comunicacao (stringReceive)")
        print (e)
        Serie.close()

def temp(txt, size):
    Fonte = pygame.font.Font("times.ttf", size)
    Texto = Fonte.render(txt, True, Black)
    screen.blit(Texto, [7, 160])

def tempTexto(txt, size):
    Fonte = pygame.font.Font("times.ttf", size)
    Texto = Fonte.render(txt, True, Black)
    screen.blit(Texto, [110, 230])

def press(txt, size):
    Fonte = pygame.font.Font("times.ttf", size)
    Texto = Fonte.render(txt, True, Black)
    screen.blit(Texto, [10, 325])

def pressTexto(txt, size):
    Fonte = pygame.font.Font("times.ttf", size)
    Texto = Fonte.render(txt, True, Black)
    screen.blit(Texto, [110, 395])

def tick(txt,size):
    Fonte = pygame.font.Font("DS-DIGI.ttf",size)
    Texto = Fonte.render(txt, True, Black)
    screen.blit(Texto, [400, 400])
```

```
s = comInit(com, baudrate)
print (s)
a = [None]*5
imprimir_1_vez = True
imprimir_cada_30sec = True
Window()
while (s != None):
    now = datetime.datetime.now()
    if( now.hour < 10):
        b = ("0" + str(now.hour))
    else:
        b = str(now.hour)
    if( now.minute < 10):
        c = ("0" + str(now.minute))

    else:
        c = str (now.minute)
    if( now.second < 10):
        d = ("0" + str(now.second))
    else:
        d = str (now.second)
    tick( b + ":" + c + ":" + d,70)
    for indice in range(5):
        a[indice] = stringReceive(s)
    if (imprimir_1_vez == True):
        Window();
        temp(a[0], 55)
        print(a[0])
        tempTexto(a[1], 70)
        print(a[1])
        press(a[2] , 55)
        print(a[2])
        pressTexto(a[3], 70)
        print(a[3])
        imprimir_1_vez = False
```

```
t1 = pygame.time.get_ticks()
    if (t1 - t0 > 30000):
        if (imprimir_cada_30sec == True):
            Window();
            temp(a[0], 55)
            print(a[0])
            pygame.draw.line(screen,(255,0,0),(30,30),(15+a[0],30),width=6)
            tempTexto(a[1], 70)
            print(a[1])
            press(a[2] , 55)
            print(a[2])
            pressTexto(a[3], 70)
            print(a[3])
            a[4] = None
            imprimir_cada_30sec = False
        for c in a:
            if c == None:
                a = [None] * 5
                t0 = t1
                imprimir_cada_30sec = True
    for event in pygame.event.get():
       if  event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()


    pygame.display.update()
    timer.tick(60)
pygame.quit()
```

Cenário de Testes

Depois dos programas estarem concluídos, tanto os programas em Arduino do display e do sensor, como do *Python* para representar a velocidade angular, passou-se para um cenário de testes onde vai ser feita a visualização dos resultados. Porém surgiram problemas no que diz respeito à implementação do sensor, nomeadamente o facto de o grupo não ter o sensor em estudo, em vez desse sensor, o grupo optou por usar outro tipo de sensor, o sensor GY-521, no qual , é um sensor com as mesmas capacidades, mas apresenta uma estrutura mais complexa em termos de implementação e programação. Visto que o código apresentado foi feito somente ao sensor L3GD20, os valores de ambos os eixos se encontram a zero, e o valor da temperatura encontra-se num valor constante.

Figura 19 - representação dos eixos e da temperatura no display

Conclusões

Com a realização deste trabalho laboratorial conseguimos entender melhor o funcionamento dos sensores, neste caso do sensor L3GD20, uma vez que este foi utilizado durante o projecto e foi implementado o código em Arduino para o correto funcionamento do mesmo.

O grupo concretizou quase todos os passos do enunciado o que possibilitou uma fácil compreensão do trabalho pedido. Inicialmente foi feito o código do sensor em Arduino, código que fazia os cálculos dos eixos que o giroscópio tem e temperatura, através do sensor fornecido. De seguida foi feita uma implementação do código do Display I2C para que se pudesse visualizar os valores lidos pelo sensor. Por fim, foi feita uma janela em *Python* utilizando a biblioteca "*pygame*" de forma a termos a melhor visualização possível dos diferentes valores da velocidade angular.

Resumindo, os objectivos foram quase todos alcançados, não foi possível realizar mais testes e verificar o correto funcionamento dos programas feitos, devido ao facto do grupo não ter a seu dispor o sensor, apesar disso, o grupo conseguiu adquirir conhecimentos acerca de toda a nova matéria leccionada.

Bibliografia

Moodle – Folhas de CF (complementares)

Moodle – Folhas de Computação Física

Moodle – DataSheet do L3GD20