Bateria virtual controlada pela Launchpad MSP430G2553

Rodrigo Magalhães Caires
FGA
Universidade de Brasília
Gama, Brasil
rodrigo_macrt@hotmail.com

Palavras-chave — MSP430, Bateria virtual.

I. Resumo

Sistema de percussão virtual controlada pela Launchpad MSP430G2553 e estimulada a partir de sensores de aceleração e giro presos ao braço ou baqueta do usuário.

II. Introdução

O conceito deste projeto é simular a experiência de tocar bateria com apenas um ou dois sensores, um microcontrolador, um bluetooth e um computador ou smartphone, ou seja, sem utilizar os componentes físicos como tambores, caixas e pratos. As vantagens deste produto estão na redução de custo, tendo em vista que um bom kit de bateria custa milhares de reais enquanto esta alternativa reduz o gasto para aproximadamente 100 reais, na flexibilidade, já que o protótipo é portátil e, portanto, de fácil montagem e deslocamento, e na comodidade, sendo capaz de reter todo som e barulho a um fone de ouvido.

III. Desenvolvimento

O prótotipo da bateria virtual utiliza um microcontrolador MSP430G2553 cujas funções principais são executar a comunicação intermediaria entre os periféricos e fazer o processamento dos dados, um sensor acelerômetro e giroscópio MPU6050, utilizado para determinar o sentido do deslocamento e fazer uma estimativa da posição relativa da “baqueta”, e um módulo bluetooth HC-05, responsável pela transmissão dos dados para um computador ou smartphone que, por fim, fará a emulação do som.

O funcionamento deste projeto se dá na captação dos movimentos dos braços ou baquetas do usuário. A captação dos dados de aceleração e giro serão executadas pelo sensor e enviadas para o microcontrolador que, por sua vez, fará a interpretação e processamento dos dados, retornando informações tangíveis como qual peça da bateria foi tocada. Essas informações serão enviadas via bluetooth para um aplicativo ou estação de aúdio com sampler de bateria, onde será reproduzido o som correspondente.

O projeto de software apresentou vários pontos problemáticos durante o desenvolvimento sendo os mais importantes a precisão dos dados durante a coleta, a propagação de erro durante as etapas de integração discreta para transformar aceleração em deslocamento e a consideração da aceleração gravitacional e a sua influência em cada eixo de acordo com o giro do sensor. As soluções utilizadas a princípio são rudimentares e para fins de teste, mas soluções mais sofisticadas foram encontradas e deverão ser aplicadas assim que as adaptações estiverem de acordo com as necessidades do projeto.

Enquanto os problemas de precisão de dados coletados e de integração foram solucionados, os

erros de estimação da gravidade nos eixos persistiram até o fim do projeto. Várias funções foram testadas na tentativa de amenizar o erro, mas nenhuma apresentou resultados eficazes. Tendo em vista o problema mencionado, algumas alterações precisaram ser feitas na proposta inicial e na meta final para que o projeto pudesse seguir em frente.

IV. Descrição de Hardware

O protótipo é composto por uma Launchpad MSP430G2553, um módulo bluetooth HC-05, um sensor acelerômetro e giroscópio MPU6050, dois resistores de 4,7 KΩ e cabos de conexão variados.

Após as devidas configurações, os pinos P1.1 e P1.2 da Launchpad podem ser utilizados como RX (Receptor) e TX (Transmissor) para a comunicação com o módulo bluetooth enquanto os pinos P1.6 e P1.7 podem ser usados como SCL (Clock serial) e SDA (Dado serial) para a comunicação com a MPU6050.

A alimentação dos periféricos é compartilhada e realizada pelos pinos VCC e GND da Launchpad. Para que a comunicação entre a Launchpad e o bluetooth funcione, a conexão dos pinos de tranmissão e recepção devem ser cruzadas, ou seja, o RX (P1.1) e TX (P1.2) da MSP430 devem ser conectados ao TX (TXD) e RX (RXD) do HC-05 respectivamente. Já para a MPU6050, a conexão de pinos é direta, ligando o pino P1.6 (SCL) ao SCL do sensor e o pino P1.7 (SDA) ao DAS do sensor.

Porém, para que ambas as comunicações funcionem corretamente, devem ser feitas algumas alterações adicionais. Para a comunicação UART, realizada entre o bluetooth e a MSP430, deve-se alterar a configuração dos jumpers TXD e RXD da vertical para a horizontal de acordo com a Figura 1 (Anexo I). Já para a comunicação I2C, realizada entre o sensor e a MSP430, deve-se remover o jumper P1.6 conectado ao LED2, vide Figura 2 (Anexo I), pois o mesmo pode afetar o funcionamento do SCL (também conectado ao pino P1.6). Além disso, devem ser adicionados dois resistores externos de pullup, um entre os SCLs e um entre os SDAs. Isso deve ser feito para regularizar o sinal que é passado entre os dois MOSFETs presentes nos terminais dos pinos I2C. A montagem completa do hardware pode ser conferida no diagrama da Figura 3 (Anexo I).

V. Descrição de Software

O programa utiliza bibliotecas especiais para facilitar a comunicação nos protocolos UART (bluetooth), I2C e para a obtenção de dados do sensor MPU6050. Faz-se também o uso das variáveis de controle ACCEL_XOUT_ANT, ACCEL_YOUT_ANT, ACCEL_ZOUT_ANT e control, sendo as três primeiras do tipo signed int e utilizadas para armazenar valores anteriores de aceleração nos eixos X, Y e Z respectivamente enquanto control é uma variável auxiliar do tipo int utilizada para controlar o comportamento de um trecho do código durante um laço de verificação.

As primeiras definições da main são correspondentes ao desligamento do Watchdog Timer e a seleção de um clock de 16 MHz. Em seguida, é chamada a função de inicialização do protocolo I2C, passando como parâmetro o endereço do sensor MPU6050 (0x68). Dentro da função de inicialização, estabelece-se a funcionalidade dos pinos P1.6 e P1.7 como SCL (Serial Clock) e SDA (Serial Data), respectivamente, através dos registradores P1SEL. Logo após, define-se um clock de 100 KHz para sincronizar a comunicação entre o mestre e o escravo e armazena-se o endereço do sensor no registrador UCB0I2CSA.

Após a inicialização do I2C, realiza-se a configuração do sensor MPU6050 por meio de outra função, definindo os endereços de acesso aos registradores responsáveis pelo armazenamento dos dados de aceleração e giro nos 3 eixos. Depois de configurar o sensor, é feita a inicialização do protocolo UART para comunicação via bluetooth. Primeiramente, define-se as funcionalidades RXD (Receiver) e TXD (Transmitter) para os pinos P1.1 e P1.2 respectivamente, novamente por meio dos registradores P1SEL. Em seguida, configura-se o

clock e a baud rate da comunicação. Para terminar a inicialização do programa, atribue-se os valores iniciais para variáveis de controle

O laço while é responsável por fazer a leitura, processamento e transmissão de dados e, a princípio, funciona indefinidamente. Primeiramente são feitas as leituras do acelerômetro pela função *Get_Accel_Values()* e do giroscópio pela função *Get_Gyro_Values()* e converte-se os valores obtidos de signed int para float. Em seguida, é chamada a função *MadgwickAHRSupdateIMU*, responsável pela definição dos quatérnions referentes a orientação do sensor e que tem os dados de aceleração e giroscópio como parâmetros de entrada. A função retorna os valores q0, q1, q2 e q3 que são a parte escalar e os 3 parâmetros da parte vetorial respectivamente. Esses dados serão usados posteriormente para determinar a influência da gravidade em cada eixo do sensor.

Os valores de aceleração e giro passam por um breve processamento envolvendo a remoção de offset e a conversão dos valores em informações mais tangíveis do tipo aceleração em g's e giro em º/s. Filtros de Kalman também são aplicados para atenuar a influência de ruídos inclusos na leitura do sensor. A influência da gravidade em cada eixo é então calculada utilizando os quaternions obtidos pelo filtro de Madgwick. Esses valores são subtraídos das acelerações medidas pelo sensor e o resultado é multiplicado por 9.8, obtendo as acelerações reais axias em m/s².Em seguida, usa-se o método da integração dupla trapezoidal para se obter o deslocamento gerado e a posição referente do sensor.

Experimentou-se dois métodos de funcionamento análogo para a estimulação da bateria: uma por posição e outra por aceleração. Resumidamente, ambos os testes comparavam os valores de posição ou de aceleração com patamares previamente estabelecidos. Quando as condições são satisfeitas, um sinal relativo à peça tocada (chimbal, caixa, bumbo, etc.) é enviado via bluetooth para o dispositivo destino. Uma função debounce também foi criada para impedir que o sinal seja enviado involuntariamente repetidas vezes.

Por fim, os valores anteriores são atualizados e a nova iteração é começada.

VI. RESULTADOS

Devido a problemas durante a estimação das gravidades axiais, não foi possível obter valores aceitáveis para a aceleração real e o método posicional se tornou inutilizável e, portanto, os resultados foram obtidos em função do método da aceleração. Sinais referentes a peça tocada são enviados quando o acelerômetro identifica uma influência de mais de 90% da aceleração gravitacional em um dos eixos. Considerando os sentidos positivos e negativos de cada eixo, um total de 6 peças puderam ser simuladas por esse método (caixa, tom 1, tom 2, bumbo, surdo e chimbal). Os sinais são enviados uma única vez via bluetooth para o receptor (computador ou smartphone) devido às funções debounce que impedem que um estado seja repetido.

Mesmo com o debounce, ainda é possível tocar uma peça consecutivas vezes sem precisar tocar outra. Basta sair da zona de ativação da peça (90% de influência gravitacional no eixo escolhido) para a zona cinza, zona onde nenhuma peça é ativada (nenhum eixo com mais de 90% de influência gravitacional) e então retornar para a zona de ativação da peça escolhida.

Os resultados foram observados a partir do software de emulação de terminais PuTTY e um exemplo das saídas pode ser observado na Figura 4 (Anexo I). A função principal do código pode ser encontrado no Anexo II.

VII. CONCLUSÃO

No geral, estre projeto conseguiu aprofundar e afixar bem os conhecimentos sobre o funcionamento dos microcontroladores, mais

especificamente quanto a lógica dos registradores e a comunicação com outros dispositivos.

Infelizmente, os resultados esperados no início deste projeto não foram inteiramente alcançados, porém a proposta inicial é possível de ser concluída. O uso de um sensor com 9 graus de liberdade (acelerômetro, giroscópio e magnetômetro) ajudaria a obter dados mais precisos quanto a orientação do sensor que, por sua vez, resultaria em estimações mais apuradas da influência da gravidade em cada eixo. Mais importante que a troca do sensor seria uma revisão mais profunda da matemática utilizada, o que deve auxiliar a entender os processos envolvidos no projeto de maneira mais clara e possivelmente auxiliar na correção de qualquer erro de cálculo que possa estar contido no código.

VIII. REFERÊNCIAS

[1] Texas Instruments, ChronosDrums. Disponível em: http://processors.wiki.ti.com/index.php/ChronosDrums

[2] Xanthium. Serial Communication Using MSP430 UART(USCI-A). Disponível em: http://xanthium.in/Serial-Communication-MSP430-UART-USCI_A

[3] SLAU144 USER GUIDE.

[4] MPU6050 DATASHEET.

[5] HC-05 DATASHEET.

[6] CHRobotics, Using accelerometers to estimate position and velocity. Disponível em: http://www.chrobotics.com/library/accel-position-velocity.

ANEXO I - Imagens

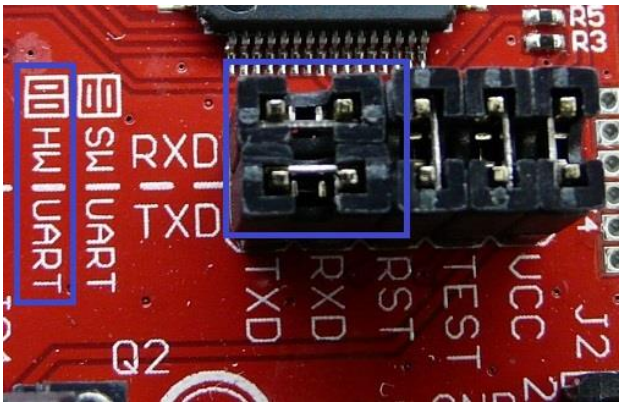

Figura 1. Jumpers UART.

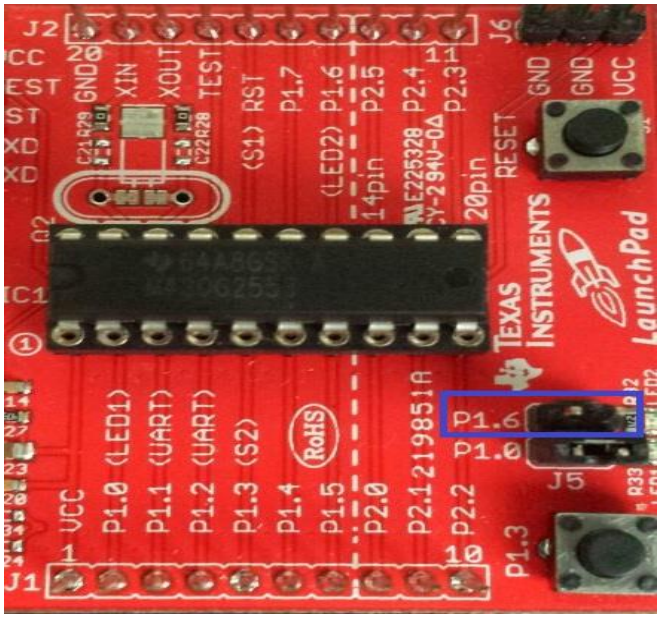

Figura 2. Jumper P1.6.

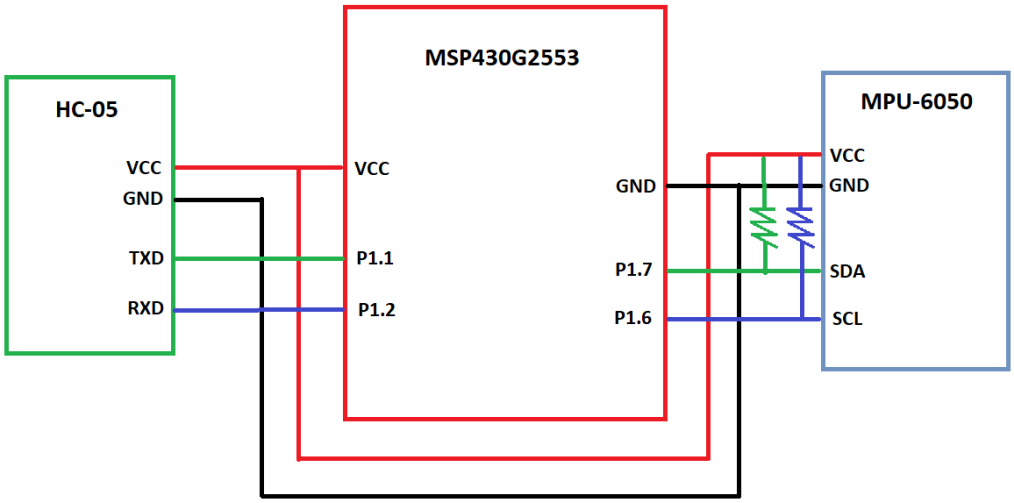

Figura 3. Montagem do hardware.

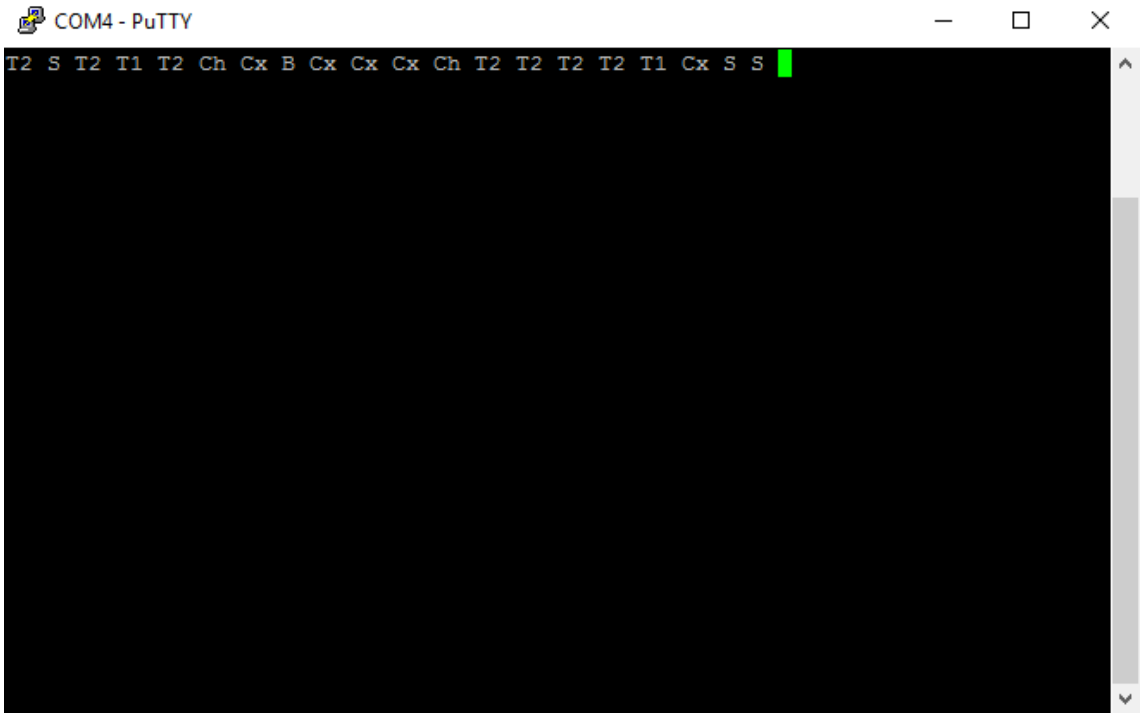

Figura 4. Teste do protótipo da bateria.

ANEXO II – Main.c

```
#define USE_I2C_USCI
#define USE_MPU6050
#define USE_UART
#define MadgwickAHRS

#include <math.h>
#include <stdlib.h>
#include "msp430g2553.h"
#include "Global.h"
#include "UART.h"
#include "I2C_USCI.h"
#include "MPU6050.h"
#include "MadgwickAHRS.h"
#include "MadgwickAHRS.c"

char buff[50];
float ACC_X, ACC_Y, ACC_Z;
float ACC_X_ANT = 0, ACC_Y_ANT = 0, ACC_Z_ANT = 0;
float GYR_X, GYR_Y, GYR_Z;
float GYR_X_ANT = 0, GYR_Y_ANT = 0, GYR_Z_ANT = 0;
float GRAV_X, GRAV_Y, GRAV_Z;
float GRAV_X_ANT = 0, GRAV_Y_ANT = 0, GRAV_Z_ANT = 0;
float RACC_X, RACC_Y, RACC_Z;
float RACC_X_ANT = 0, RACC_Y_ANT = 0, RACC_Z_ANT = 0;
float dx, dy, dz;
float dxa = 0, dya = 0, dza = 0;
float px, py, pz;

int xp = 0, xn = 0, yp = 0, yn = 0, zp = 0, zn = 0;
```

```
void main()
{
        WDTCTL = WDTPW | WDTHOLD;
    BCSCTL1 = CALBC1_16MHZ;
        DCOCTL = CALDCO_16MHZ;
    BCSCTL1 |= XT2OFF;
        BCSCTL2 = SELM_0;
        __delay_cycles(100000);
    P1DIR = BIT0;

    Init_I2C_USCI(MPU6050_ADDRESS); //1.7 SDA, 1.6 SCL
        Setup_MPU6050(); // +-2g, +-250º/s, 1KHz srf accel

    __delay_cycles(320000);

    UART_Init();          //P1.1 RXD, P1.2 TXD

    while(1)
    {
         Get_Accel_Values();
         Get_Gyro_Values();

         ACC_X = (float)ACCEL_XOUT;
         ACC_Y = (float)ACCEL_YOUT;
         ACC_Z = (float)ACCEL_ZOUT;

         GYR_X = (float)GYRO_XOUT;
         GYR_Y = (float)GYRO_YOUT;
         GYR_Z = (float)GYRO_ZOUT;

         //ESTIMAÇÃO DE ORIENTAÇÃO (QUATERNIONS) - MADGWICK
```

```
MadgwickAHRSupdateIMU(GYR_X, GYR_Y, GYR_Z, ACC_X, ACC_Y, ACC_Z);

//OFFSET GIRO - DATASHEET
if(abs(GYR_X) <= 2620)
  GYR_X = 0;
else{
  if(GYR_X >= 0)
    GYR_X -= 1310;
  else
    GYR_X += 1310;
}

if(abs(GYR_Y) <= 2620)
  GYR_Y = 0;
else{
  if(GYR_Y >= 0)
    GYR_Y -= 1310;
  else
    GYR_Y += 1310;
}

if(abs(GYR_Z) <= 2620)
  GYR_Z = 0;
else{
  if(GYR_Z >= 0)
    GYR_Z -= 1310;
  else
    GYR_Z += 1310;
}

//OFFSET ACELERAÇÃO - DATASHEET
```

```
if(abs(ACC_X) <= 819.2)
  ACC_X = 0;
else{
  if(ACC_X >= 0)
    ACC_X -= 409.6;
  else
    ACC_X += 409.6;
}

if(abs(ACC_Y) <= 819.2)
  ACC_Y = 0;
else{
  if(ACC_Y >= 0)
    ACC_Y -= 409.6;
  else
    ACC_Y += 409.6;
}

/*if(abs(ACC_Z) <= 819.2)
  ACC_Z = 0;
else{
  if(ACC_Z >= 0)
    ACC_Z -= 409.6;
  else
    ACC_Z += 409.6;
}*/

if(abs(ACC_Z) <= 1310.7)
  ACC_Z = 0;
else{
  if(ACC_Z >= 0)
```

```
    ACC_Z -= 655.36;
   else
    ACC_Z += 655.36;
  }

  //GIRO EM º/S : (+-250 -> 131, +-500 -> 66.5, +-1000 -> 32.8, +-2000 -> 16.4) - DATASHEET
  GYR_X_ANT /= (131/1);
  GYR_Y_ANT /= (131/1);
  GYR_Z_ANT /= (131/1);

  GYR_X /= (131/1);
  GYR_Y /= (131/1);
  GYR_Z /= (131/1);

  //ACELERAÇÃO EM G : (+-2g -> 16384, +-4g -> 8192, +-8g -> 4096, +-16g -> 2048) - DATASHEET
  ACC_X_ANT /= (16384/1);
  ACC_Y_ANT /= (16384/1);
  ACC_Z_ANT /= (16384/1);

  ACC_X /= (16384/1);
  ACC_Y /= (16384/1);
  ACC_Z /= (16384/1);

  /*ACC_X *= 10;
  ACC_Y *= 10;
  ACC_Z *= 10;*/

  //FILTRO KALMAN
  ACC_X = 0.9*ACC_X_ANT + 0.1*ACC_X;
  ACC_Y = 0.9*ACC_Y_ANT + 0.1*ACC_Y;
```

```
ACC_Z = 0.9*ACC_Z_ANT + 0.1*ACC_Z;

GYR_X = 0.9*GYR_X_ANT + 0.1*GYR_X;
GYR_Y = 0.9*GYR_Y_ANT + 0.1*GYR_Y;
GYR_Z = 0.9*GYR_Z_ANT + 0.1*GYR_Z;

//INFLUENCIA DA GRAVIDADE NOS EIXOS - FABIO VARESANO
//q0 - ESCALAR; q1, q2, q3 - VETORIAL
GRAV_X = (2*(q1*q3 - q0*q2));
GRAV_Y = (2*(q0*q1 + q2*q3));
GRAV_Z = (q0*q0 - q1*q1 - q2*q2 + q3*q3);

/*GRAV_X *= 10;
GRAV_Y *= 10;
GRAV_Z *= 10;*/

//KALMAN GRAV
GRAV_X = 0.9*GRAV_X_ANT + 0.1*GRAV_X;
GRAV_Y = 0.9*GRAV_Y_ANT + 0.1*GRAV_Y;
GRAV_Z = 0.9*GRAV_Z_ANT + 0.1*GRAV_Z;

//ACELERAÇÃO REAL NOS EIXOS - FABIO VARESANO
RACC_X = (ACC_X - GRAV_X)*9.8;
RACC_Y = (ACC_Y - GRAV_Y)*9.8;
RACC_Z = (ACC_Z - GRAV_Z)*9.8;

//KALMAN ACELERAÇÃO REAL
RACC_X = 0.9*RACC_X_ANT + 0.1*RACC_X;
RACC_Y = 0.9*RACC_Y_ANT + 0.1*RACC_Y;
RACC_Z = 0.9*RACC_Z_ANT + 0.1*RACC_Z;
```

```
//DESLOCAMENTO EM CM - NÃO UTILIZADO
dx = RACC_X*0.3*0.3*100;
dy = RACC_Y*0.3*0.3*100;
dz = RACC_Z*0.3*0.3*100;

//POSIÇÃO - NÃO UTILIZADO
px += dx;
py += dy;
pz += dz;

//VERIFICAÇÃO - ACELERAÇÃO
if(10*ACC_X < 0.9){
  xp = 0;
}

if(10*ACC_X > -0.9){
  xn = 0;
}

if(10*ACC_Y < 0.9){
  yp = 0;
}

if(10*ACC_Y > -0.9){
  yn = 0;
}

if(10*ACC_Z < 0.9){
  zp = 0;
}
```

```
if(10*ACC_Z > -0.9){
  zn = 0;
}

//IMPRESSÃO DE RESULTADOS - POR ACELERAÇÃO
if(10*ACC_X > 0.9 && xp == 0){        //X POS
  UART_Write_Char('T');               //TOM 2
  UART_Write_Char('2');
  UART_Write_Char(' ');
  xp = 1;
}

if(10*ACC_X < -0.9 && xn == 0){       //X NEG
  UART_Write_Char('C');               //CAIXA
  UART_Write_Char('x');
  UART_Write_Char(' ');
  xn = 1;
}

if(10*ACC_Y > 0.9 && yp == 0){        //Y POS
  UART_Write_Char('T');               //TOM 1
  UART_Write_Char('1');
  UART_Write_Char(' ');
  yp = 1;
}

if(10*ACC_Y < -0.9 && yn == 0){       //Y NEG
  UART_Write_Char('S');               //SURDO
  UART_Write_Char(' ');
  yn = 1;
}
```

```
            if(10*ACC_Z > 0.9 && zp == 0){        //Z POS
              UART_Write_Char('C');               //CHIMBAL
              UART_Write_Char('h');
              UART_Write_Char(' ');
              zp = 1;
            }

            if(10*ACC_Z < -0.9 && zn == 0){       //Z NEG
              UART_Write_Char('B');               //BUMBO
              UART_Write_Char(' ');
              zn = 1;
            }

            //ATUALIZÇÃO DOS DADOS ANTERIORES
             ACC_X_ANT = ACC_X;
             ACC_Y_ANT = ACC_Y;
             ACC_Z_ANT = ACC_Z;

             GYR_X_ANT = GYR_X;
             GYR_Y_ANT = GYR_Y;
             GYR_Z_ANT = GYR_Z;

             dxa = dx;
             dya = dy;
             dza = dz;
        }

}
```