Detector de Queda de Idosos

Projeto Final

Ingrid Miranda de Sousa

Programa de Engenharia Eletrônica
Faculdade Gama - Universidade de Brasília
ingridmsousa0@gmail.com

Yago Uriel Fernandez da Silva

Programa de Engenharia Eletrônica
Faculdade Gama - Universidade de Brasília
yagouriel1@gmail.com

I. Resumo

A falta de acompanhamento e cuidado de idosos é um problema comum enfrentado atualmente. Por essa razão, o projeto visa projetar e implementar um sistema capaz de monitorar e detectar constantemente a movimentação de pessoas idosas, e assim, ser capaz de distinguir casos em que haja a queda. Para isso, utiliza-se o acelerômetro, a fim de verificar as variações na posição do indivíduo, e o módulo bluetooth, que enviará os dados obtidos para um celular que posteriormente poderá requisitar ajuda para um número pré-programado.

Palavras-chave: Cuidado de idosos, queda, acelerômetro, módulo bluetooth.

II. Introdução

Devido a maior expectativa de vida o número de pessoas com mais de 65 anos no Brasil deve praticamente quadruplicar até 2060, segundo dados obtidos pela pesquisa realizada pelo IBGE [1]. Com esse aumento precisaremos cada vez mais de atenção e cuidados com a população de idosos.

Observando a necessidade de prevenção e visando obter melhor cuidado dentre as pessoas acima de 65 anos de idade, foi realizado uma pesquisa e encontrado em um documento publicado com autoria da sociedade brasileira de Geriatria e Gerontologia que a queda é o mais sério e frequente acidente doméstico que ocorre com idosos e a principal etimologia de morte acidental. A estimativa da incidência de quedas por faixa etária é de 28% a 35% nos idosos com mais de 65 anos e 32% a 42% naqueles com mais de 75 anos [2].

Vemos então que a prevenção da queda e prestação de um socorro rápido pode ser de grande importância para redução de sequelas, síndromes, traumas e mortalidade causadas à pessoas idosas. E assim, partindo dessa análise, levantamos o objetivo principal do nosso projeto.

III. Desenvolvimento

O nosso projeto tem como objetivo principal a criação de um sistema capaz de monitorar e detectar, em tempo real, casos que ocorram a queda de pessoas idosas.

Tendo em vista gerar maior qualidade de vida para o idoso e para seus familiares observando que, na maioria dos casos de queda em idosos, o medo é a consequência mais recorrente. Isso faz com que parte dessas pessoas abandonem certas atividades, e outras passem a quase não se locomoverem [3]. Desse modo, a supervisão nesses casos, em tempo real, são fundamentais para promover maior cuidado com a saúde do idoso.

Um atendimento veloz pode amenizar o risco de se agravarem as lesões ocasionadas pela queda, que incluem: transtornos psicológicos, fraturas, redução da capacidade funcional e da independência, até a morte [3].

Esse sistema trará então mais segurança, privacidade, mobilidade, conforto e independência à esses indivíduos, para que possam praticar suas atividades normalmente, pois saberão que caso necessitem de ajuda, serão atendidos de forma rápida e segura [4].

A. Descrição do hardware

O Hardware do sistema (Figura 1) consiste em uma combinação de um microcontrolador com sensores com o objetivo de poder identificar a queda de uma pessoa idosa. Tal sistema foi dividido em 2 blocos:

Bloco 1 - O bloco 1 é responsável por receber, trabalhar e controlar as informações analógicas e digitais utilizando como hardware principal para essa função o

microcontrolador MSP430, juntamente com os sensores acelerômetro e giroscópio (MPU6050) que são responsáveis pela identificação do momento da queda.

O MSP430 (Modelo utilizado no projeto: MSP430G2553IN20) é um microcontrolador RISC de 16 Bits voltado para aplicações de baixo consumo de energia e é fabricado pela Texas Instrument.

O acelerômetro e giroscópio (Modelo utilizado no projeto: MPU6050) possui de forma integrada no mesmo módulo, acelerômetro, giroscópio e termômetro monitora as variações na posição e aceleração dos eixos x, y e z da pessoa idosa e permite avaliar quando mudado bruscamente os valores da posição inicial, indicando a queda.

O botão localizado em posição de fácil acesso que permita ser acionado em uma situação de emergência pelo usuário.

Bloco 2 - O bloco 2 é responsável pela transmissão da informação de queda do microcontrolador para o celular do usuário que então realizará o envio da mensagem para o celular de um contato de emergência utilizando o aplicativo criado em seu celular. O módulo bluetooth (Modelo utilizado no projeto: HC-05) realizará a conexão de dados entre o sistema e um aparelho exterior permitindo assim, em caso de queda, o contato via mensagem com uma pessoa responsável pelo idoso.

Tal sistema deverá ser implementado em um cinto, que permite a identificação facilitada da posição inicial da pessoa idosa, além de gerar maior conforto para quem o usa.

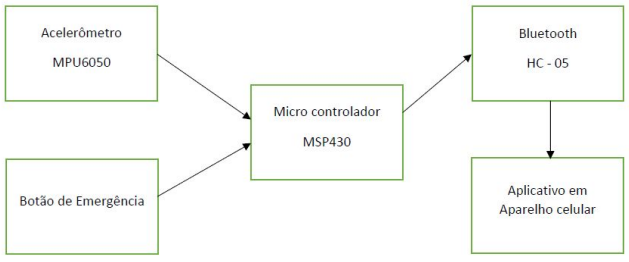

Figura 1: Hardware do Sistema

B. Descrição do Software

O software utilizado neste projeto inicialmente foi a IDE Energia devida à maior simplicidade de codificação. Essa IDE, cedida pela Texas Instrument, é uma plataforma de prototipagem eletrônica de código aberto com objetivo de trazer o framework Arduino para o launchpad do MSP430 facilitando assim a sua programação.

Após o entendimento do protótipo utilizando a IDE Energia foi feita a migração e adequação dos códigos para a IDE Code Composer Studio(**CCS**) que é projetado principalmente para o desenvolvimento de sistemas embarcados e de baixo nível (BareMetal) com depuração baseada em JTAG. Tal software exige um maior entendimento de codificação.

Para a transmissão da mensagem de alerta ao usuário foi criado um software utilizando outro programa chamado AppInventor, tal programa é um criador de aplicativos open source que permite o desenvolvimento em blocos de aplicativos. Sua estrutura pode ser vista na Fig. 2.

O aplicativo criado contém um botão com opção de conectar o bluetooth do aparelho celular ao módulo HC-05. E caso o MPU 6050 apresente uma mudança muito brusca, ou seja, uma condição de queda, envia uma mensagem de texto pré-determinada para um número de emergência.

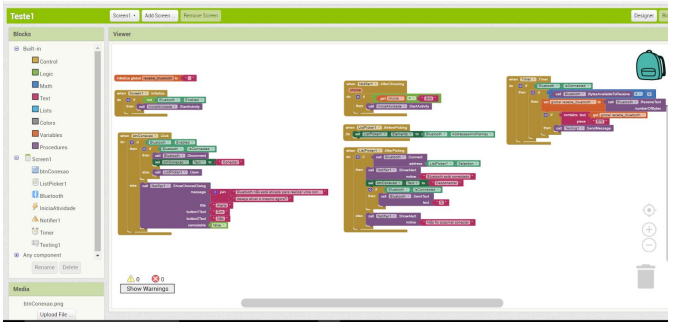

Figura 2 - Estrutura de blocos para a criação do aplicativo

Os códigos da implementação do projeto tanto no software energia quanto no CCS se encontram nos anexos.

O código de implementação do módulo MPU 6050 na IDE Energia (**Anexo 1**) consiste na conexão utilizando o protocolo de comunicação I2C do módulo bluetooth com o MSP430. No MSP430 os pinos referentes a essa conexão são o P1.6 (Conexão clock I2C - SCL) e o P1.7 (Conexão de dados - SDA). Esse código tem a função de iniciar a transmissão de dados entre os dispositivos, acorda o MPU 6050, receber os dados dos eixos x,y e z do acelerômetro, giroscópio e a temperatura e armazená-los em registradores.

Para a conexão do módulo bluetooth HC05 conectou-se os pinos TX e RX ao P1.1 e P1.2 do MSP430, respectivamente. E utilizou-se o pino P1.5 para receber o sinal do MPU 6050. O código de implementação deste módulo consiste na transmissão de dados na forma serial, por meio da comunicação UART.

IV. Resultados

Para validar os valores adquiridos no módulo MPU 6050 foi utilizado as expressions do CCS, que permite visualizar os valores armazenados nos registradores utilizados

no codigo.

E para a comprovação da conexão do módulo bluetooth utilizou-se o aplicativo Serial Bluetooth, por meio de um celular Android. Assim, foi indicado no terminal os valores recebidos do acelerômetro em tempo real.

Por fim, para certificar a eficácia do projeto usou-se o aplicativo criado para a disciplina, de modo que em condições simuladas de queda, foi possível receber uma mensagem de texto. Assim, como é possível analisar nas Figs. 3 e 4.

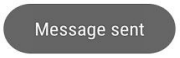

Figura 3 - Captura de tela do aplicativo ao enviar SMS

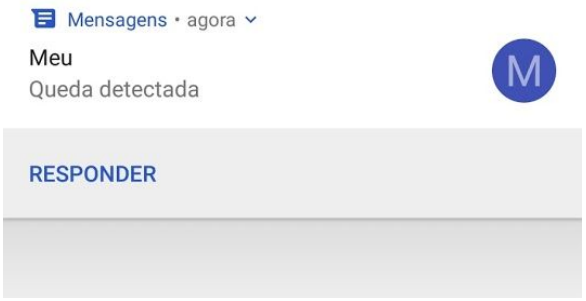

Figura 4 - Mensagem recebida

V. Conclusões

O objetivo principal do projeto era a construção de um sistema capaz de monitorar e detectar a queda de uma pessoa idosa. Tal sistema, no modelo de protótipo, se mostrou eficiente na aquisição e envio em tempo real dos dados recebidos do acelerômetro e também no envio de forma rápida de mensagem com o alerta de queda para o celular de emergência cadastrado pelo usuário.

Já no modelo final, utilizando o software CCS, pode-se observar diversos entraves na codificação e obtenção dos resultados. Fatos esses devido a dificuldade de criação de um padrão para identificação da queda como também o problema encontrado no acelerômetro que para ser reconhecido pelo software necessitava ser desligado e ligado diversas vezes durante o processo.

Os problemas não conseguiram ser solucionados a tempo para a apresentação do projeto final levando assim a um protótipo que não estava calibrado da forma correta para serem realizados testes da sua utilização e eficiência.

Verificamos que é necessário o aprimoramento futuro do projeto para que o mesmo possa concluir com efetividade a sua função e de fato melhor a qualidade de vida de pessoas idosas.

VI. Referências Bibliográficas

[1] "Projeção do IBGE 2000 -2060" , http://www.ibge.gov.br/home/estatistica/populacao/projecao_da_populacao/2013/default_tab.shtm (Acesso em 2 de abril de 2017).

[2] "Documento da Sociedade brasileira de geriatria e de gerontologia", http://sbgg.org.br/wp-content/uploads/2014/10/queda-idosos.pdf (Acesso em 2 de abril de 2017).

[3] LIMA, D. P. Queda nos idosos: fatores de risco e implicações. Fortaleza, 2005.

[4] LEMOS, A. A. Detector de Quedas de Idosos. Curitiba, 2011.

[5] "Software Energia", http://energia.nu/ (Acesso em 5 de maio de 2017).

ANEXO 1

Código de implementação do módulo MPU 6050 na IDE Energia

```
#include<Wire.h>

const int MPU_addr=0x68;  // Endereço I2C do MPU- 6050
int16_t AcX,AcY,AcZ,Tmp,GyX,GyY,GyZ;
void setup(){

 Wire.setModule(0);

  Wire.begin();
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x6B);
  Wire.write(0);     // set to zero (Acorda o MPU-6050)
  Wire.endTransmission(true);
  Serial.begin(9600);
}
void loop(){
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x3B);  // Começando com o registrador 0x3B
(ACCEL_XOUT_H)
  Wire.endTransmission(false);
  Wire.requestFrom(MPU_addr,14,true);
AcX=Wire.read()<<8|Wire.read();  // 0x3B
(ACCEL_XOUT_H) & 0x3C (ACCEL_XOUT_L)
  AcY=Wire.read()<<8|Wire.read();  // 0x3D
(ACCEL_YOUT_H) & 0x3E (ACCEL_YOUT_L)
  AcZ=Wire.read()<<8|Wire.read();  // 0x3F
(ACCEL_ZOUT_H) & 0x40 (ACCEL_ZOUT_L)
  Tmp=Wire.read()<<8|Wire.read();  // 0x41 (TEMP_OUT_H)
& 0x42 (TEMP_OUT_L)
  GyX=Wire.read()<<8|Wire.read();  // 0x43
(GYRO_XOUT_H) & 0x44 (GYRO_XOUT_L)
  GyY=Wire.read()<<8|Wire.read();  // 0x45
(GYRO_YOUT_H) & 0x46 (GYRO_YOUT_L)
  GyZ=Wire.read()<<8|Wire.read();  // 0x47
(GYRO_ZOUT_H) & 0x48 (GYRO_ZOUT_L)
  Serial.print("AcX = "); Serial.print(AcX);
  Serial.print(" | AcY = "); Serial.print(AcY);
  Serial.print(" | AcZ = "); Serial.print(AcZ);
  Serial.print(" | Tmp = "); Serial.print(Tmp/340.00+36.53);
//Equação de graus C do datasheet
  Serial.print(" | GyX = "); Serial.print(GyX);
  Serial.print(" | GyY = "); Serial.print(GyY);
  Serial.print(" | GyZ = "); Serial.println(GyZ);
  delay(333);
}
```

ANEXO 2

Código de implementação do módulo HC05 na IDE Energia

```
void setup() {
  // inicializar comunicação serial com 9600 bits por segundo:
  Serial.begin(9600); //
}

void loop() {
  // leitura do pino analógico A3 - potênciometro
  int sensorValue = analogRead(A3);
  // printar o valor lido
  Serial.println(sensorValue);
  delay(10); // delay entre as leituras para estabilidade
}
```

ANEXO 3

Código de implementação do módulo HC05 no Code Composer

```
#include <msp430.h>
void print(char *texto){
    unsigned int i = 0;
    while(texto[i] != '\0'){
        while (!(IFG2&UCA0TXIFG));
        UCA0TXBUF = texto[i];      // TX -> Recebe caracter + 1
        i++;
}}

void printNumber(unsigned int num){
    char buf[6];
    char *str = &buf[5];
    *str = '\0';
    do {
        unsigned long m = num;
        num /= 10;
        char c = (m - 10 * num) + '0';
        *--str = c;
}
   while(num);
    print(str);
}

void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;    // Parar Watchdog
    if (CALBC1_1MHZ==0xFF){
        while(1);
    }
```

```
   DCOCTL = 0;  // Selecionar configurações de DCO mais
lentas
   BCSCTL1 = CALBC1_1MHZ;    // Setar DCO para 1 MHz
   DCOCTL = CALDCO_1MHZ;

   P1SEL = BIT1 + BIT2 ;   // Selecionar a função UART
RX/TX nos pinos P1.1 e P1.2
   P1SEL2 = BIT1 + BIT2;

   UCA0CTL1 |= UCSSEL_2;      // UART Clock -> SMCLK
   UCA0BR0 = 104;        // Setar Baud Rate para 1MHz 9600
   UCA0BR1 = 0;
   UCA0MCTL = UCBRS_1;
   UCA0CTL1 &= ~UCSWRST;   // Inicializar módulo
UART

   ADC10AE0 |= BIT5;      // Selecionar pino P1.5 para opção
ADC
   ADC10CTL1 = INCH_5;             // Canal ADC -> 1 (P1.1)
   ADC10CTL0 = SREF_0 + ADC10SHT_3 + ADC10ON;  //
Ref -> Vcc, 64 CLK S&H

   while(1)
   {
      ADC10CTL0 |= ENC + ADC10SC;                // Início da
amostragem e conversão
      while(ADC10CTL1 & ADC10BUSY);  // Esperar o final
da conversão
      unsigned int adcVal = ADC10MEM;    // Leitura do valor
ADC
      printNumber(adcVal);               // Printar valor no
UART
      print("\r\n");                              // Printar nova linha
      __delay_cycles(10000);
   }
}
```

Anexo 4

Codigo do CCS Modulo 1

```
#include <msp430g2553.h>

unsigned char RX_Data[6];
unsigned char TX_Data[2];
unsigned char RX_ByteCtr;
unsigned char TX_ByteCtr;
unsigned char slaveAddress = 0x68;
int xAccel;
int yAccel;


long map(long Accel, long in_min, long in_max, long
out_min, long out_max)
{
 return (Accel - in_min) * (out_max - out_min) / (in_max -
in_min) + out_min;
}

void i2cInit(void)
{
   // set up I2C module
   UCB0CTL1 |= UCSWRST;                 // Enable SW reset
   UCB0CTL0 = UCMST + UCMODE_3 + UCSYNC;   //
I2C Master, synchronous mode
   UCB0CTL1 = UCSSEL_2 + UCSWRST;         // Use
SMCLK, keep SW reset
   UCB0BR0 = 10;                        // fSCL = SMCLK/12 =
~100kHz
   UCB0BR1 = 0;
   UCB0CTL1 &= ~UCSWRST;                // Clear SW reset,
resume operation
}

void i2cWrite(unsigned char address)
{
   //__disable_interrupt();
   IE2 &= ~UCB0TXIE;                // Desabilita o interrupt
   UCB0I2CSA = address;             // Load slave address
   IE2 |= UCB0TXIE;                 // Enable TX interrupt
   while(UCB0CTL1 & UCTXSTP);         // Ensure stop
condition sent
   UCB0CTL1 |= UCTR + UCTXSTT;        // TX mode and
START condition
   //__bis_SR_register(GIE);
   //while((IFG2 & UCB0TXIFG) == 0);
   __bis_SR_register( LPM0_bits + GIE);       // sleep until
UCB0TXIFG is set ... }
}
void i2cRead(unsigned char address)
{
   //__disable_interrupt();
   IE2 &= ~UCB0RXIE;                // desabilita o interrupt
   UCB0I2CSA = address;             // Load slave address
   IE2 |= UCB0RXIE;                 // Enable RX interrupt
   while(UCB0CTL1 & UCTXSTP);         // Ensure stop
condition sent
   UCB0CTL1 &= ~UCTR;               // RX mode
   UCB0CTL1 |= UCTXSTT;              // Start Condition
   //__bis_SR_register(GIE);
   //while((IFG2 & UCB0RXIFG) == 0);
   __bis_SR_register(LPM0_bits + GIE);       // sleep until
UCB0RXIFG is set ...
}



int main(void){
   BCSCTL1 = CALBC1_1MHZ;  // seleciona o Master clock
de 1Mhz
   DCOCTL = CALDCO_1MHZ;   // seleciona o clock de
1Mhz
```

```
  P1DIR |= 0x21;
  P1OUT &= 0x00;

  P1SEL |= BIT6 + BIT7;                 // Assign I2C pins to
USCI_B0
  P1SEL2|= BIT6 + BIT7;                 // Assign I2C pins to
USCI_B0


  i2cInit();
  slaveAddress = 0x68;                  // MPU-6050 address
  TX_Data[1] = 0x6B;                    // address of
PWR_MGMT_1 register
  TX_Data[0] = 0x00;                    // set register to zero
(wakes up the MPU-6050)
  TX_ByteCtr = 2;
  i2cWrite(slaveAddress);

  for(;;){

    //P1OUT |= 0x21;


    // Point to the ACCEL_ZOUT_H register in the
MPU-6050
    slaveAddress = 0x68;                // MPU-6050 address
    TX_Data[0] = 0x3B;                  // register address
    TX_ByteCtr = 1;
    i2cWrite(slaveAddress);

    // Read the two bytes of data and store them in zAccel
    slaveAddress = 0x68;                // MPU-6050 address
    RX_ByteCtr = 6;
    i2cRead(slaveAddress);

    xAccel  = RX_Data[5] << 8;          // MSB
    xAccel |= RX_Data[4];               // LSB
    yAccel  = RX_Data[3] << 8;          // MSB
    yAccel |= RX_Data[2];               // LSB

    //zAccel  = RX_Data[1] << 8;          // MSB
    //zAccel |= RX_Data[0];

    if(xAccel < 0)
      P1OUT|= 0x21;
    //while(xAccel < 0);


    //left =
map(xAccel,0,20000,limite_medio,limite_superior);
    // right =
map(xAccel,0,-20000,limite_medio,limite_superior);
    //up =
map(yAccel,0,20000,limite_medio,limite_superior);
    // down =
map(yAccel,0,-20000,limite_medio,limite_superior);
  }
}

#pragma vector = USCIAB0TX_VECTOR

__interrupt void USCIAB0TX_ISR(void)
{

  if(UCB0CTL1 & UCTR)               // TX mode (UCTR ==
1)
  {
    if (TX_ByteCtr)                 // TRUE if more bytes
remain
    {
      TX_ByteCtr--;                 // Decrement TX byte counter
      UCB0TXBUF = TX_Data[TX_ByteCtr];    // Load TX
buffer
    }
    else                    // no more bytes to send
    {
      UCB0CTL1 |= UCTXSTP;          // I2C stop condition
      IFG2 &= ~UCB0TXIFG;       // Clear USCI_B0 TX
int flag

      __bic_SR_register_on_exit(CPUOFF);  // Exit LPM0

    }
  }
  else // (UCTR == 0)               // RX mode
  {
    RX_ByteCtr--;                   // Decrement RX byte
counter
    if (RX_ByteCtr)                 // RxByteCtr != 0
    {
      RX_Data[RX_ByteCtr] = UCB0RXBUF;    // Get
received byte
      if (RX_ByteCtr == 1)          // Only one byte left?
      UCB0CTL1 |= UCTXSTP;          // Generate I2C stop
condition
    }
    else                    // RxByteCtr == 0
    {
      RX_Data[RX_ByteCtr] = UCB0RXBUF;    // Get final
received byte
      __bic_SR_register_on_exit(CPUOFF);  // Exit LPM0
    }
  }
}
```