RELATÓRIO FINAL
Módulo Eletrônico Para Bicicletas

Fábio Barbosa Pinto
Programa de Engenharia Eletrônica
Faculdade do Gama
Universidade de Brasília – UnB
Brasília, Brasil
fabio_bbarbosa@hotmail.com

João Pedro Moreira da Silva
Programa de Engenharia Eletrônica
Faculdade do Gama
Universidade de Brasília – UnB
Brasília, Brasil
jptekc@gmail.com

Larissa Aidê Araújo Rocha
Programa de Engenharia Eletrônica
Faculdade do Gama
Universidade de Brasília – UnB
Brasília, Brasil
larissa.aide@hotmail.com

***RESUMO*— O projeto visa desenvolver um módulo eletrônico para bicicletas, que tem como objetivos: rastrear a localização da bicicleta; e demostrar a intenção de movimento do ciclista por meio de um sistema de setas. O produto desenvolvido exibe as coordenadas da localização da bicicleta a partir de mensagens enviadas ao celular do ciclista. Além disso, irá indicar, através de setas, a intenção do ciclista de mudar de direção na via.**

Palavras-chave— rastreador; bicicleta; sinalização; MSP430.

I. JUSTIFICATIVA

Com o excesso de carros nas ruas e lentidão no trânsito, cada vez mais pessoas buscam meios de transporte alternativos. A preferida é a bicicleta, que promove uma grande qualidade de vida. Mas infelizmente, o aumento da violência urbana e acidentes é mais rápido que uma busca pela sustentabilidade.

Apesar de ser um ótimo transporte alternativo, sabe-se que a bicicleta não é um investimento barato, podendo custar até 35 mil reais dependendo do modelo ou marca. Independente do valor encontrado no mercado, certamente, se o proprietário usa sua bicicleta com frequência, ficar sem ela fará muita falta.

De acordo com o Cadastro Nacional de Bicicletas Roubadas [1], dentre os 30 municípios que divulgaram as ocorrências de furtos e roubos nos últimos seis anos, as quatro cidades que lideram o ranking são respectivamente: São Paulo, Rio de Janeiro, Curitiba e Brasília. Nesse período, foram cerca de 1.940 casos, sem contar com as ocorrências que não são registradas.

Além da preocupação com o furto de bicicletas, é imprescindível zelar também pela segurança do ciclista. Notícias sobre acidentes envolvendo bicicletas são comuns no Brasil. Dados de 2014, mostram que 1.357 ciclistas morreram vítimas de acidentes de trânsito no Brasil, além disso, em 2016, ocorreram 11.741 internações de ciclistas vítimas de acidentes [2]. De acordo com o Departamento Nacional de Infraestrutura de Transportes (DNIT) [3] só no ano de 2011 foram 1.698 casos de acidentes envolvendo ciclistas. Sendo que 246, equivalente a 14.5%, acabaram na morte.

Há diversos modelos no mercado de rastreadores para veículos, o site Vox Popi [4] fez uma lista com mais de 17 modelos, onde mostra a diferença entre os produtos.

O diferencial deste projeto é ir além de ser um mero rastreador, dando importância também a segurança do ciclista enquanto trafega em vias com outros veículos. Assim, o módulo eletrônico para bicicletas poderá torna-se o investimento ideal para aumentar as chances de recuperá-la, caso seja roubada e provê maior segurança para o ciclista no trânsito.

II. OBJETIVO

Construir um módulo eletrônico capaz de transmitir as coordenadas geográficas de uma bicicleta através de mensagens, via celular, e implementar um sistema para indicar a intenção do ciclista de mudar de direção na via.

III. REQUISITOS

Para o correto funcionamento do módulo eletrônico, será necessária uma leitura das coordenadas geográficas da bicicleta, através de um GPS. Os dados recebidos serão enviados via SMS, através do módulo GSM em conjunto com um chip SIM. É preciso que o usuário possua um número de celular para cadastrar e utilizar as funcionalidades do rastreador. Para o sistema de setas, será necessária uma matriz de LEDs e uma chave seletora.

Os dados e comandos recebidos serão interpretados através do microcontrolador MSP430, esse dispositivo possui uma poderosa CPU RISC de 16 bits, que alia uma boa eficiência com um baixo consumo de energia. Além disso, será imprescindível uma bateria recarregável.

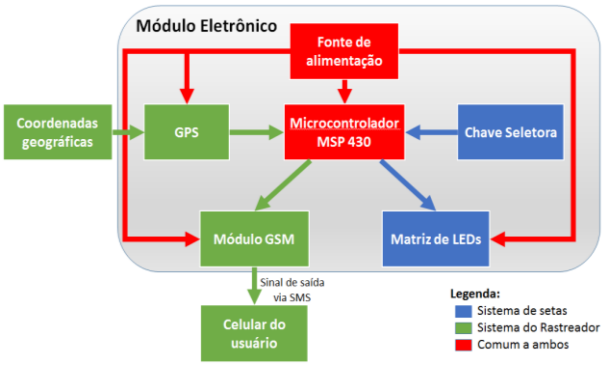

Figura 1. Diagrama de blocos do módulo eletrônico para bicicletas.

Na Figura 1, através do diagrama de blocos do módulo eletrônico, é possível ver com maior facilidade os componentes eletrônicos e suas interações descritas neste tópico.

IV. BENEFÍCIOS

Por meio do módulo eletrônico, o usuário poderá localizar sua bicicleta em qualquer lugar, caso a mesma tenha sido roubada ou furtada e trará maior segurança ao ciclista, ao trafegar em vias públicas.

O rastreador seria uma contraproposta ao uso do seguro de bicicletas, já que, geralmente, eles não cobrem bicicletas desprotegidas, ou seja, sem a presença do seu proprietário. Se você deixar a sua bicicleta presa ao poste ou esquecer no parque e a levaram, infelizmente, você não terá mais como recuperá-la se optar apenas pelo seguro.

O sistema de setas alerta a intenção dos atos do ciclista para outros condutores, visando garantir maior segurança ao trafegar em vias públicas.

Dessa forma, o proprietário da bicicleta terá uma segurança maior em deixá-la presa em algum lugar, como postes, árvores ou paraciclos e até mesmo ao andar com ela pela rua.

V. HARDWARE

Os materiais utilizados até o ponto de controle 3 foram:

- 2 MSP-EXP430G2553LP
- 1 módulo GSM800L
- 1 módulo GPS GY-NEO6M
- 1 módulo Matriz LED 8x8 com MAX7219
- 3 chaves seletoras
- Jumpers
- 1 Bateria alcalina (9V)
- 1 Módulo de alimentação MB102 3.3V/5V
- 1 Regulador de tensão LM2596
- 1 Carregador portátil de celular Power Bank Pineng Pn-952

O Hardware do projeto é constituído por dois microcontroladores, na qual utilizou-se o MSP430G2553IN20, que é um microcontrolador RISC de 16 bits, voltado para aplicações de baixo consumo de energia, fabricado pela Texas Instruments. Os microcontroladores foram combinados com os módulos GPS, GSM e o módulo matriz LED.

Visando uma melhoria no projeto, o sistema foi dividido em duas partes: Parte 1 (sistema de setas), parte 2 (rastreador).

Parte 1: Essa parte é responsável pela sinalização da bicicleta, onde será integrado o módulo LED 8x8 MAX7219 ao MSP430.

A matriz LED pode ser conduzida de duas maneiras (paralela ou serial). Aqui o conduzimos de maneira serial, para simplificar a utilização e reduzir o tamanho do circuito. O CI MAX7219 é um driver de exibição de entrada /saída de cátodo comum de série, que interage com microcontroladores como arduinos, MSP430, Raspberry e outros, para um LED numérico de 7 segmentos de até 8 dígitos, exibições de gráfico de barras ou 64 LEDs individuais.

Na figura 5.1 é possível observar o modo como o circuito foi montado já com as entradas e saídas do microcontrolador definidas.

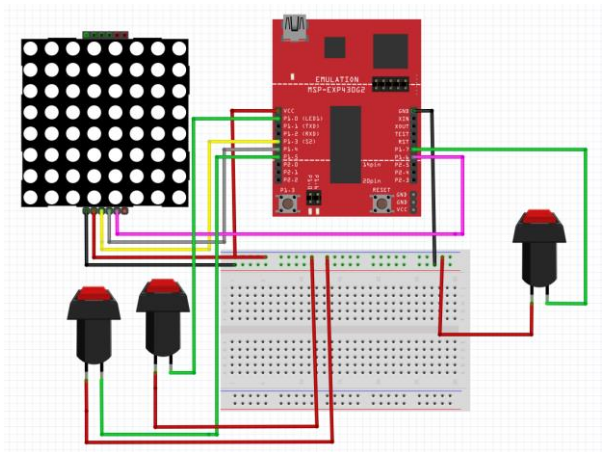
Figura 2. Circuito montado para a sinalização da bicicleta

Para a primeira parte do projeto o circuito foi alimentado com uma bateria de 9V através do módulo de alimentação MB102 3.3V/5V

Figura 3. Alimentação referente ao circuito das setas de sinalização

Parte 2: Esta parte é responsável pela transmissão de informação da localização da bicicleta. Um microcontrolador receberá as informações provenientes do módulo GPS, e os comandos do módulo GSM, através de SMS.

Para o microcontrolador, utilizou-se outra placa MSP430G2553IN20, que recebe as informações do módulo GPS (GPS NEO6M), através da comunicação UART. Este mesmo módulo recebe o comando proveniente do módulo GSM SIM800L, através de SMS, e envia a localização da bicicleta, também utilizando comunicação UART.

Para a segunda parte do projeto é necessário fornecer uma corrente maior para o circuito, já que pelas especificações do datasheet do módulo GSM SIM800l o

mesmo pode chegar a consumir até 2 A de corrente (Em consulta a alguns fóruns de dúvidas, foram relatados casos em que o módulo consumiu mais de 2 A de corrente). Dessa forma o circuito foi alimentado por carregador portátil de celular Power Bank Pineng Pn-952, que fornece até 2.1 A de corrente e um por um regulador de tensão LM2596.

Figura 4. Alimentação do segundo circuito do projeto

VI. CÓDIGO

Inicialmente no projeto o software utilizado foi a IDE Energia, devida à simplicidade na codificação, que é baseada no Wiring e na linguagem Processing, a mesma base da conhecida IDE do Arduino. Posteriormente, foi realizada a adequação dos códigos para o software Code Composer Studio (CCS), que é utilizado para o desenvolvimento de sistemas embarcados e de baixo nível de consumo de energia.

A. Módulo GSM

O código gerado para a comunicação da placa MSP430 com módulo GSM800l utiliza o software Energia. O intuito do código é enviar uma SMS para o destinatário ao qual o número de celular foi adicionado ao próprio código (futuramente o corpo da mensagem será a localização da bicicleta fornecida pelo módulo GPS).

A comunicação entre o Microcontrolador e o módulo GSM é feita através da porta Serial (UART), dessa forma uma vez que as comunicações tenham sido declaradas e estabelecidas, basta somente enviar comandos AT do datasheet para que o módulo possa processá-los e agir de acordo.

Um fator importante analisado é que módulo pode utilizar muita corrente de sua fonte em picos de transmissão (de até 3A e devido a isso não se pode alimentá-lo diretamente na MSP), então embora o código funcione perfeitamente se a fonte não for suficiente para alimentá-lo, ele irá reiniciar aleatoriamente durante as transmissões.

Em fase de teste a cada comando enviado ao módulo GSM, normalmente, retorna-se uma resposta, assim sempre que se envia um comando, é necessário esperar a resposta para saber se os dados enviados foram realmente enviados. Após enviarmos os comandos AT de inicialização, o módulo respondeu, porém o mesmo não enviou a mensagem.

Por meio de pesquisas realizadas em materiais, fóruns e espaços para discussão, disponibilizados e indicados pela Texas Instruments, observamos tal fato pode ter ocorrido devido ao sinal do GSM, que apresenta uma variação na frequência em que o LED pisca, quando encontra rede, e poucas vezes foi possível percebê-la.

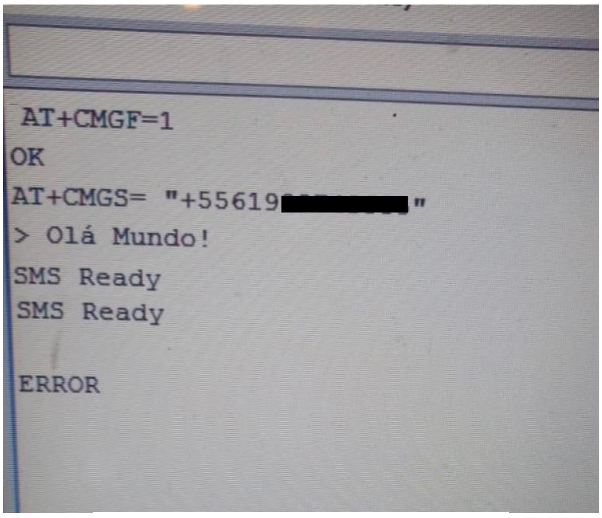

Figura 5. Resposta dada pelo módulo GSM

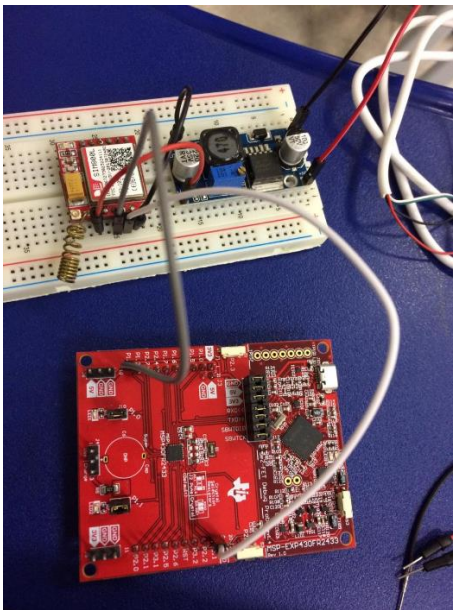

Figura 6. Montagem do circuito com o módulo GSM para teste

B. Módulo GPS

Para realizar a montagem do projeto, deve-se conectar os pinos TX e RX do módulo GPS aos pinos P1.1 e P1.2 respectivamente na MSP430. Caso os jumpers estejam conectados na forma Hradware Serial, é necessário que os pinos TX do módulo esteja no P1.2 e o pino RX esteja no pino conectado ao P1.1. É possível utilizar o próprio microcontrolador para alimentar o módulo GPS, deve-se conectar os cabos Vcc e Gnd desse aos respectivos do MSP430.

O código gerado para a comunicação da placa MSP430 com módulo GPS GY-NEO6M da uBlox utiliza o software Energia. De maneira geral o código simplesmente lê as strings originais do módulo GPS e mostra na tela serial. Ele também usa uma biblioteca para tratar os dados e usar funções pré-definidas para obter apenas as informações desejadas. Essa biblioteca é a A TinyGPS++, que permite extrair as informações fornecidas pelo módulo GPS, fornecendo apenas as informações desejadas.

Devido a complexibilidade do módulo, o mesmo retorna a localização monitorada na forma de um link para o Google Maps, e envia a mesma para o responsável em forma de SMS pelo módulo GSM. O módulo utiliza os dados da matriz de satélites da NMEA (National Marine Electronics

Association) para coordenadas, o software separa esses valores e retorna o que é pedido, latitude, longitude, altitude, hora. Os dados fornecidos pela NMEA no formato *raw* é mostrado na figura a seguir.

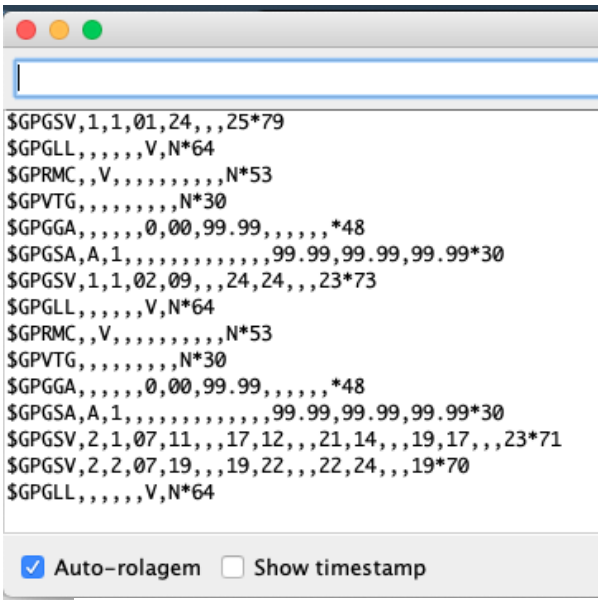

Figura 7. Dados fornecidos pela NMEA de forma *raw*.

Cada sigla tem o seu significado:

$GPBOD - Traçando origem ao destino.
$GPBWC - Calculando distância ao destino desejado.
$GPGGA - Global Positioning System Fix Data.
$GPGLL - Posição geográfica, latitude / longitude.
$GPGSA - GPS DOP e satelites ativos.
$GPGSV - GPS Satelites a vista.
$GPR00 - List of waypoints in currently active route
$GPRMA - Recommended minimum specific Loran-C data
$GPRTE – Rotas
$GPVTG - Localização válida.
$GPWPL - Localização.
$GPZDA - Data & Hora.

Usando a biblioteca TinyGPS++, ela decodifica essas sentenças nas coordenadas desejadas no formato [LATITUDE, LONGITUDE], como mostra a figura a seguir.

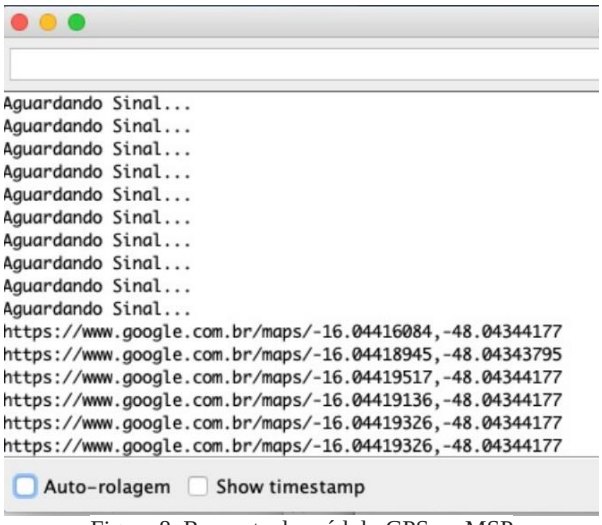

Figura 8. Resposta do módulo GPS no MSP

Para que o GPS e o GSM retornem um link para o Google Maps, foi necessário a criação de uma string que concatena, uma string inicial para o link, e os valores obtidos para latitude e longitude. Para o cálculo da distância de um ponto a outro utilizou-se funções já prontas para a biblioteca do módulo GPS. O uso do link do Google Maps se dá ao fato dos dados da NMEA estarem desatualizados.

Para que módulo GPS responda leva cerca de 20 a 30 segundos até obter o sinal, e é recomendado céu aberto para pleno funcionamento.

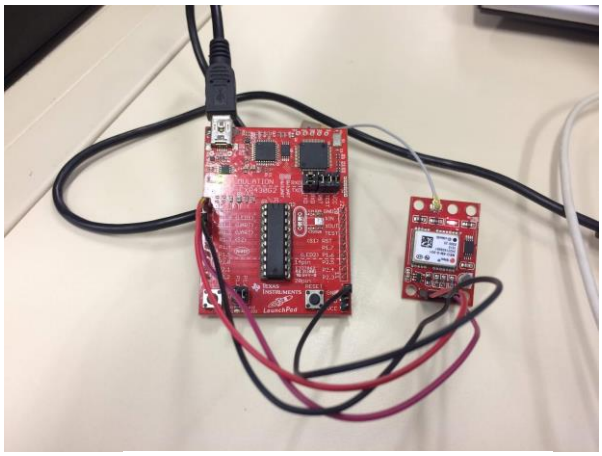

Figura 9. Módulo GPS conectado a MSP430

Para a migração do código do módulo para o software CCS, foi criado a biblioteca *uart* que faz a comunicação serial do módulo GPS com a placa. Na *main* do código foram criadas as funções Time (referente a hora), Lat (referente a latitude), Long (referente a longitude) e Date (referente a data), para receber os dados do GPS.

Na main também contém um loop infinito com uma sequência de if encadeados que realiza uma espécie de Bit Start da comunicação assíncrona. Em seguida os dados são agrupados strncpy myGPS que copia os primeiros caracteres da origem para o destino.

```
for(;;){
    temp_char = uart_getc();  //Coleta dados do GPS
    if (temp_char == '$') {
        temp_char = uart_getc();
        if (temp_char == 'G') {
            temp_char = uart_getc();
            if (temp_char == 'P') {
                temp_char = uart_getc();
                if (temp_char == 'R') {
                    temp_char = uart_getc();
                    if (temp_char == 'M') {
                        temp_char = uart_getc();
                        if (temp_char == 'C') {
                            //Código para salvar partes específicas do GPS DATA Stream
                            uart_gets(temp_sentence, RMC_SENT_LEN);
                            strncpy(myGPS->Time, (temp_sentence), 11);
                            strncpy(myGPS->Lat, (temp_sentence + 13), 10);
                            strncpy(myGPS->Long, (temp_sentence + 25), 11);
                            strncpy(myGPS->Date, (temp_sentence + 50), 7);
                            return;
```

Figura 10. Parte responsável por coletar dados do GPS

Os dados obtidos do código no CCS, referente ao GPS, não forneceram as informações exatas da localização. Uma solução para se obter as coordenadas com exatidão um tratamento com operações matemáticas e os dados do GPS teriam que ser feitas, porém devido a complexidade do módulo e o tempo até a entrega do trabalho não foi possível realizar tal operação.

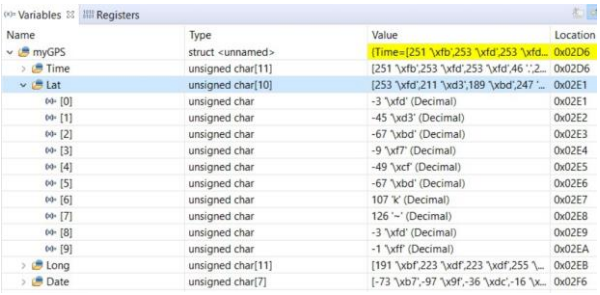

Figura 11. Dados extraídos do GPS no CCS

C. Código de integração dos módulos GPS e GSM

Para a parte 2 do projeto tanto o módulo GPS como o módulo GSM utilizam comunicação UART, dessa forma foi necessária a criação de uma nova porta UART para um dos módulos, sendo escolhido o módulo GPS, pois o mesmo apenas envia dados para o microcontrolador, ou seja, só utiliza uma porta RX, enquanto o módulo GSM recebe e envia dados para o microcontrolador, utilizando uma porta RX e uma TX. Para que o GPS e o GSM retornem um link para o Google Maps, foi necessário a criação de uma string que concatena, uma string inicial para o link, e os valores obtidos para latitude e longitude. O uso do link do Google Maps se dá ao fato dos dados da NMEA estarem desatualizados.

O código de integração dos módulos foi realizado no Energia IDE e segue em anexo no final do projeto. Porém devido a complexidade dos módulos, principalmente do módulo GSM SIM800l, não foi possível a migração do código de integração para o CCS.

D. Setas de Sinalização

A execução do código gera 4 desenhos possíveis: seta para a esquerda, seta para a direita, parando e o de sinalização (que são todos os LEDs do módulo piscando).

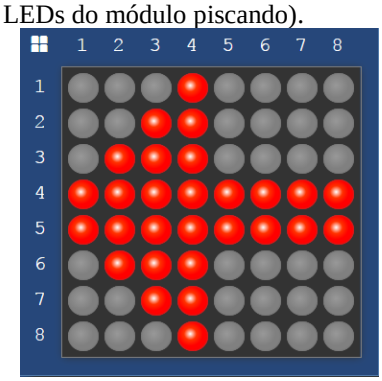

Figura 12. Desenho de referência da seta para esquerda

A função no código foi descrita da seguinte maneira:

```
// BITS SETADOS EM CADA LINHA P/ FORMAR A SETA P/ ESQUERDA
/*
  B00010000,
  B00110000,
  B01110000,
  B11111111,
  B11111111,
  B01110000,
  B00110000,
  B00010000
 */
void seta_esquerda(){

   led8x8(0x18,0x3c,0x7e,0xff,0x18,0x18,0x18,0x18);
}
```

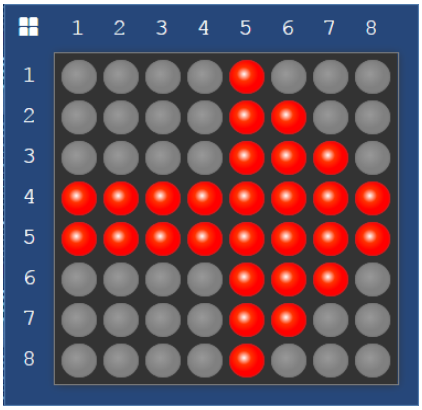

Figura 13. Desenho de referência da seta para direita

A função no código foi descrita da seguinte maneira:

```
//BITS SETADOS EM CADA LINHA P/ FORMAR A SETA P/ DIREITA
/*
  B00001000,
  B00001100,
  B00001110,
  B11111111,
  B11111111,
  B00001110,
  B00001100,
  B00001000
 */

// FUNÇAO SETA P/ DIREITA

void seta_direita(){

    led8x8(0x18,0x3c,0x7e,0xff,0x18,0x18,0x18,0x18);
```

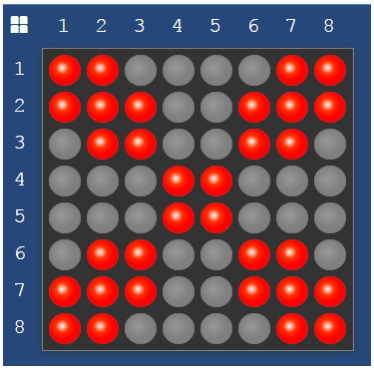

Figura 14. Desenha de referência para parando

A função no código foi descrita da seguinte maneira:

```
void stop(){

   led8x8(0xc3,0xe7,0x66,0x18,0x18,0x66,0xe7,0xc3);
}
```

E por fim a sinalização, onde todos os bits da matriz foram setados, seguidos de um delay para que ficasse piscando.

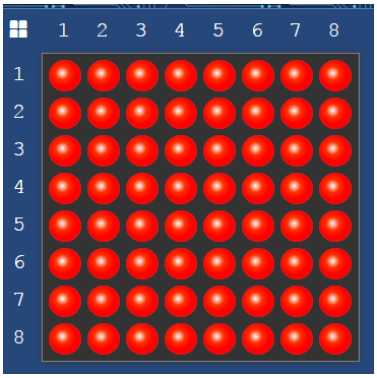

Figura 15. Desenho de referência para sinalização

A função no código foi descrita da seguinte maneira:

```
void sinalizaçao(){
write8x8(0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff);
delay (10000);
```

VII. RESULTADOS E DISCUSSÕES

A parte 1 do projeto, sistema de setas, foi concluída com êxito. Conforme os resultados esperados, as setas para a esquerda, direita, indicação de parando e a sinalização foram mostradas na matriz de LED, de acordo com os comandos enviados, como pode ser visto na Figura 16.

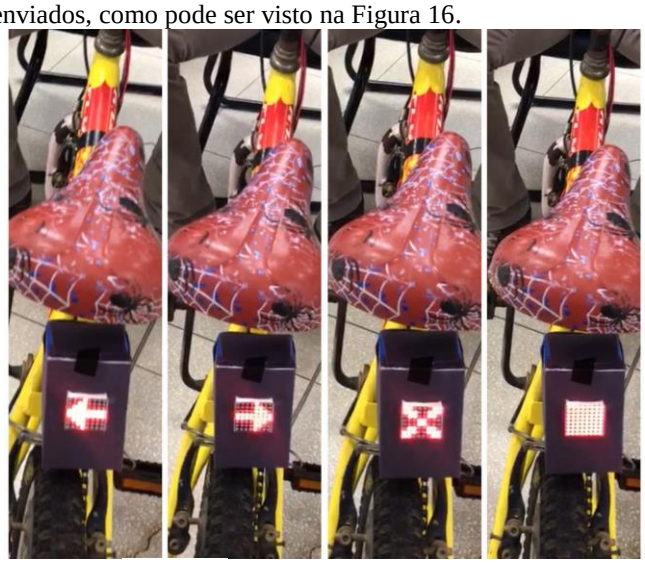
Figura 16. Sistema de setas implementado

Já a parte 2 do projeto, o rastreador, ficou incompleta de acordo com a proposta inicial.

A primeira dificuldade com o módulo GSM foi quanto a alimentação, sendo sanada posteriormente com o uso de um regulador de tensão e um carregador de celular portátil. Outra dificuldade foi quanto ao sinal do GSM, o módulo do GSM apresenta uma variação na frequência que o LED pisca, quando encontra rede, e poucas vezes foi possível percebê-la.

Com o módulo GPS NEO6M, a dificuldade de encontrar rede também foi averiguada. Através de testes feitos no CCS e no IDE foi possível fazer o GPS procurar rede, mas não retornou saídas satisfatórias por conta da falta de sinal.

Em ambos os módulos, os códigos foram compilados, mas não obtiveram saídas aceitáveis para a conclusão do projeto.

VIII. CONSIDERAÇÕES FINAIS

A realização deste projeto foi agradável, oferecendo oportunidade de aprendizagem e construção de novos conhecimentos ainda não explorados pela dupla de alunos.

Muitas dificuldades foram encontradas durante a construção do projeto, desde ligar o MSP e conseguir fazê-lo executar um simples trecho de código em uma plataforma de programação qualquer até a montagem de circuitos e programação de funções específicas do MSP.

Apesar do objetivo geral deste projeto não ter sido plenamente alcançado, os conhecimentos adquiridos durante a execução do projeto foram importantes para consolidar o aprendizado das aulas expositivas realizadas, bem como adquirir experiência prática na construção de circuitos e desenvolvimento de código em microcontroladores.

Como lições aprendidas, destaca-se que o planejamento é importante para o sucesso do projeto, sobretudo a execução do planejamento é ainda mais importante para a aprendizagem.

Ressalta-se também que o projeto deve ser melhor estimado pelos alunos, para evitar o descumprimento do escopo definido no início do projeto.

IX. REFERÊNCIAS

[1] Exame, “Segundo dados, furtos e roubos de bicicleta aumentam no Brasil”. Disponível em: https://exame.abril.com.br/negocios/dino/segundo-dados-furtos-e-roubos-de-bicicleta-aumentam-no-brasil-seguradora-e-alternativa/ < Acesso em: 04/09/2018>

[2] G1, “Brasil tem em média 32 ciclistas internados por dia devido a acidentes” Dsponível em: http://g1.globo.com/bom-diabrasil/noticia/2017/03/brasil-tem-em-media-32-ciclistas-internados-pordia-devido-acidentes.html <Acesso em: 01/10/2018>

[3] DNIT, “Número de vitimados envolvidos por tipo de usuário”, 2011

[4] Vox Popi, “Top rastreadores GPS GSM para bike e carros”. Disponível em: http://voxpopi.blogspot.com/2018/02/comparacao-rastreadores-veiculos.html <Acesso em: 01/10/2018>

[5] Como Proteger sua Bike Contra Roubo http://www.revistabicicleta.com.br/bicicleta.php?como_proteger_sua_bike_contra_roubo&id=46230866 <Acesso em: 04/09/2018>

[6] Câmeras Flagram Roubo de Bicicletas https://g1.globo.com/df/distrito-federal/noticia/cameras-de-seguranca-flagram-roubo-de-bicicletas-de-dentro-de-predio-na-asa-norte.ghtml <Acesso em: 04/09/2018>

[7] GSM o que é e como funciona https://www.oficinadanet.com.br/artigo/733/gsm_o_que_e_e_como_funciona <Acesso em: 04/09/2018

X. ANEXOS

A. *Código GSM*

```
#include <SoftwareSerial.h>
 // Incluimos a livraria SoftwareSerial
SoftwareSerial mySerial(1, 2);  // Declaramos os pinos RX(1) y TX(2) que vamos a usar

void setup(){
Serial.begin(9600);       // Iniciamos a comunicação serial
mySerial.begin(9600);     // Iniciamos uma segunda comunicação serial
delay(1000);              // Pausa de 1 segundo
EnviaSMS();               // Chamada a função que envia o SMS
}

void loop(){
if (mySerial.available()){          // Se a comunicação SoftwareSerial tem dados
  Serial.write(mySerial.read());    // Obtemos por comunicação serie normal
}

if (Serial.available()){            // Se a comunicação serie normal tem dados
  while(Serial.available()) {       // e enquanto tenha dados para mostrar
    mySerial.write(Serial.read());  // Obtemos pela comunicação SoftwareSerial
  }
  mySerial.println();               // Enviamos um fim de linha
}
}

// Função para o envio de um SMS
void EnviaSMS(){
 mySerial.println("AT+CMGF=1");               // Ativamos a função de envio de SMS
 delay(100);                                 // Pequena pausa
 mySerial.println("AT+CMGS=\"+5561995--------\"");  // Definimos o número do destinatário em formato internacional
 delay(100);                                 // Pequena pausa
 mySerial.print("Comunicaçao GSM + MSP430");    // Definimos o corpo da mensagem
 delay(500);                                 // Pequena pausa
 mySerial.print(char(26));                        // Enviamos o equivalente a Control+Z
 delay(100);                                 // Pequena pausa
 mySerial.println("");                          // Enviamos um fim de linha
 delay(100);                                 // Pequena pausa
}
```

B. Código GPS CCS

```
#include "msp430g2553.h"
#include <string.h>
#include "uart.h"

#define RMC_SENT_LEN 57
#define BTN BIT3
```

```
#define LED BIT0
#define RXD BIT1
#define TXD BIT2



volatile unsigned int tx_flag;
volatile unsigned char tx_char;
volatile unsigned int rx_flag;
volatile unsigned char rx_char;


void uart_init(void)
{
    P1SEL = RXD + TXD;                  //Setup the I/O
    P1SEL2 = RXD + TXD;
   P1DIR |= LED;                       //P1.0 red LED. Toggle when char received.
   P1OUT |= LED;                        //LED off
    UCA0CTL1 |= UCSSEL_2;                  //SMCLK

                                    //8,000,000Hz, 9600Baud, UCBRx=52, UCBRSx=0, UCBRFx=1

    UCA0BR0 = 52;                    //8MHz, OSC16, 9600
    UCA0BR1 = 0;                    //((8MHz/9600)/16) = 52.08333
    UCA0MCTL = 0x10|UCOS16;            //UCBRFx=1,UCBRSx=0, UCOS16=1
    UCA0CTL1 &= ~UCSWRST;                  //USCI state machine
    IE2 |= UCA0RXIE;                   //Enable USCI_A0 RX interrupt

    rx_flag = 0;                    //Set rx_flag to 0
    tx_flag = 0;                    //Set tx_flag to 0
    return;

}


unsigned char uart_getc()

{
    while (rx_flag == 0);
    rx_flag = 0;
   return rx_char;
}

void uart_gets(char* Array, int length)

{
    unsigned int i = 0;
    while((i < length))

    {

        Array[i] = uart_getc();
        if(Array[i] == '\r')

        {
```

```
            for( ; i < length ; i++)
            {
                Array[i] = '\0';
            }
            break;
        }
        i++;
    }

   return;

}

void uart_putc(unsigned char c)
{

    tx_char = c;

    IE2 |= UCA0TXIE;

    while(tx_flag == 1);

    tx_flag = 1;

    return;

}

void uart_puts(char *str)
{
    while(*str) uart_putc(*str++);
    return;
}

#pragma vector = USCIAB0TX_VECTOR
__interrupt void USCI0TX_ISR(void)
{
    UCA0TXBUF = tx_char;
    tx_flag = 0;
    IE2 &= ~UCA0TXIE;
}

#pragma vector = USCIAB0RX_VECTOR          .

__interrupt void USCI0RX_ISR(void)

{
    rx_char = UCA0RXBUF;
    rx_flag = 1;
    P1OUT ^= LED;

}


typedef struct{
```

```
    char Time[11];
    char Lat[10];
    char Long[11];
    char Date[7];
} gps;

void get_gps(gps *myGPS);

int main(void){
  gps myGPS;

  WDTCTL = WDTPW + WDTHOLD;// STOP WATCHDOG TIMER

  BCSCTL1 = CALBC1_8MHZ;// FAZ A FREQUÊNCIA DE ACORDO COM O LAUNCHPAD 8MHZ
  DCOCTL = CALDCO_8MHZ;

  P1DIR &= ~BTN;
  P1REN |= BTN;
  P1OUT |= BTN;

  uart_init();
  __enable_interrupt();

  while(1){
    if((P1IN&BTN)==0){
      __delay_cycles(100000);
      get_gps(&myGPS);
    }
  }
}

void get_gps(gps *myGPS){
    char temp_sentence[RMC_SENT_LEN];
    char temp_char;

    for(;;){
        temp_char = uart_getc();  //Coleta dados do GPS
        if (temp_char == '$') {
            temp_char = uart_getc();
            if (temp_char == 'G') {
                temp_char = uart_getc();
                if (temp_char == 'P') {
                    temp_char = uart_getc();
                    if (temp_char == 'R') {
                        temp_char = uart_getc();
                        if (temp_char == 'M') {
                            temp_char = uart_getc();
                            if (temp_char == 'C') {
                                //Código para salvar partes específicas do GPS DATA Stream
                                uart_gets(temp_sentence, RMC_SENT_LEN);
                                strncpy(myGPS->Time, (temp_sentence), 11);
                                strncpy(myGPS->Lat, (temp_sentence + 13), 10);
                                strncpy(myGPS->Long, (temp_sentence + 25), 11);
                                strncpy(myGPS->Date, (temp_sentence + 50), 7);
                                return;
```

```
                    }
                  }
                }
              }
            }
          }
        }
      }
```

C. Codigo GPS ENERGIA IDE

```
#include <TinyGPS++.h>
/* Cria um objeto chamado gps da classe TinyGPSPlus */
TinyGPSPlus gps;

volatile float minutes, seconds;
volatile int degree, secs, mins;

void setup() {
  Serial.begin(9600);          /* Define a taxa de transmissão para comunicação serial */
}

void loop() {
       smartDelay(1000);     /* Gerar atraso preciso de 1ms */
       if (!gps.location.isValid())
       {
         Serial.print("Latitude : ");
         Serial.println("*****");
         Serial.print("Longitude : ");
         Serial.println("*****");
       }
       else
       {
         //DegMinSec(gps.location.lat());
         Serial.print("Latitude em graus decimais : ");
         Serial.println(gps.location.lat(), 6);
//         Serial.print("Latitude em graus minutos em segundos : ");
//         Serial.print(degree);
//         Serial.print("\t");
//         Serial.print(mins);
//         Serial.print("\t");
//         Serial.println(secs);
         //DegMinSec(gps.location.lng());        /* Converte o valor do grau decimal em graus minutos, segundo formulário */
         Serial.print("Longitude em decimais degraus : ");
         Serial.println(gps.location.lng(), 6);
//         Serial.print("Longitude em degraus minutos segundos : ");
//         Serial.print(degree);
//         Serial.print("\t");
//         Serial.print(mins);
//         Serial.print("\t");
//         Serial.println(secs);
       }
       if (!gps.altitude.isValid())
```

```
      {
        Serial.print("Altitude : ");
        Serial.println("*****");
      }
      else
      {
        Serial.print("Altitude : ");
        Serial.println(gps.altitude.meters(), 6);
      }
      if (!gps.time.isValid())
      {
        Serial.print("Time : ");
        Serial.println("*****");
      }
      else
      {
        char time_string[32];
        sprintf(time_string, "Time : %02d/%02d/%02d \n",gps.time.hour(), gps.time.minute(), gps.time.second());
        Serial.print(time_string);
      }
}

static void smartDelay(unsigned long ms)
{
  unsigned long start = millis();
  do
  {
    while (Serial.available())          /* Codifica dados lidos do GPS enquanto os dados estão disponíveis na porta serial  */
      gps.encode(Serial.read());
            /* Encode basicamente é usado para analisar a cadeia de caracteres recebida pelo GPS e armazená-la em um
buffer para que as informações possam ser extraídas dela */
  } while (millis() - start < ms);
}

//void DegMinSec( double tot_val)           /* Converte dados em graus decimais em graus minutos segundos */
//{
// degree = (int) tot_val;
// minutos = tot_val - degree;
// segundos = 60 * minutos;
// minutos = (int) segundos;
// mins = (int) minutos;
// segundos = segundos - minutos;
// segundos = 60 * segundos;
// segundos = (int) segundos;
//
```

D. Código Módulo Matriz de LED 8x8

1.1 Main

```
#include "max7219.h"
#include "comando.h"
#include "figura.h"
#include <msp430g2553.h>
```

```
int main(){

  WDTCTL = WDTPW + WDTHOLD;   // Desabilita WDT
  DCOCTL = CALDCO_1MHZ;     // 1 Mhz DCO
  BCSCTL1 = CALBC1_1MHZ;

  limpa_tela();

 setIntensidade(0xff);
  setTestMode(0);
  setdesliga(0);
  showdigit(8);

  conf_botao();

  __enable_interrupt();

   initialise();  // Funçao definida em modulo que habilita as entradas CS, DIN e CLK do módulo

//CRIAR LAÇO EM QUE O PRIMEIRO ESTADO DOS LEDS SEJA A SINALIZAÇÃO
            while(1){
    sinalizaçao();
  }

}

// interrupção que habilita o display com o botão

#pragma vector=PORT1_VECTOR
__interrupt void Port_1(void){

  if((P1IN & BTN_DIREITA)==0){
      seta_direita();
    }else if((P1IN & BTN_ESQUERDA)==0){
      seta_esquerda();
    }else if((P1IN & STOP_BTN)==0){
      stop();

    }else{
       P1IFG = 0;
    }

}
```

1.2 Comando

```
#include "comando.h"
#include "max7219.h"
#include <msp430g2553.h>

void delay(volatile unsigned int i){
  while((i--)>0);
}
```

```
// limpa o display
void limpa_display(){
  write8x8(0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0);
}

// Funçao que configura os botoes
void conf_botao(){
  P1DIR &= ~(BTN_DIREITA + BTN_ESQUERDA + STOP_BTN); // configura os botoes como entrada
  P1OUT &= ~(BTN_DIREITA + BTN_ESQUERDA + STOP_BTN; //Desliga ambos os leds
  P1IE |= (BTN_DIREITA + BTN_ESQUERDA + STOP_BTN);
  P1IFG &= ~(BTN_DIREITA + BTN_ESQUERDA + STOP_BTN);
  P1REN = (BTN_DIREITA + BTN_ESQUERDA + STOP_BTN);
  P1IES |= (BTN_DIREITA + BTN_ESQUERDA + STOP_BTN);
}
```

1.3 MAX7219

```
#include "max7219.h"
#include "comando.h"
#include <msp430g2553.h>

#define MAX7219_DIN BIT3
#define MAX7219_CS  BIT4
#define MAX7219_CLK BIT5


//Laço responsavel em fazer a comunicaçao do clk do modulo
static void MAX7219_SendByte (unsigned char dataout)
{
  char i;
  for (i=8; i>0; i--) {
    unsigned char mask = 1 << (i - 1);
    P1OUT &= ~(MAX7219_CLK);
    if (dataout & mask)
      P1OUT |= MAX7219_DIN;
    else
      P1OUT &= ~(MAX7219_DIN);
    P1OUT |= MAX7219_CLK;
  }
}

// Funçao que Leva o pino CS para nivel alto e seta os pinos CS, DIN, CLK para a saida

void initialise(){
  P1OUT |= MAX7219_CS;
  P1DIR |= MAX7219_DIN;
  P1DIR |= MAX7219_CS;
  P1DIR |= MAX7219_CLK;
  output(0x0b, 7);
  output(0x09, 0x00);
}

void output(char address, char data){
  P1OUT |= MAX7219_CS;
```

```
  MAX7219_SendByte(address);
  MAX7219_SendByte(data);
  P1OUT &= ~(MAX7219_CS);
  P1OUT |= MAX7219_CS;
}

//funçao modo teste
void setTestMode(int on){
  output(0x0f, on ? 0x01 : 0x00);
}

void setdesliga(int off){
  output(0x0c, off ? 0x00 : 0x01);
}

void showdigits(char numdigits){
  output(0x0b, numdigits-1);
}


// funçao brilho do modulo

void setIntensidade(char brightness){
  output(0x0a, brightness);
}

// Funçao que leva DIN para alto (DIN é utilizado para inserir os bits)
void put_byte(char data) {
  char i = 8;
  char mask;
  while(i > 0) {
    mask = 0x01 << (i - 1);
    P1OUT &= ~(MAX7219_CLK);
    if (data & mask){
      P1OUT |= MAX7219_DIN;
    }else{
      P1OUT &= ~(MAX7219_DIN);
    }
    P1OUT |= MAX7219_CLK;
    --i;
  }
}

//Define a matriz de LED (registrador, coluna)
void max_single(char reg, char col) {
  P1OUT &= ~(MAX7219_CS);
  put_byte(reg);
  //asm("mov.w reg, R15");
  //asm("call #putByte");
  //asm("pop R15");
  put_byte(col);
  P1OUT &= ~(MAX7219_CS);
  P1OUT |= (MAX7219_CS);
}
```

```
//Configuraçao da matriz
void led8x8(char a, char b, char c, char d, char e, char f, char g, char h){
  max_single(1,a);
  max_single(2,b);
  max_single(3,c);
  max_single(4,d);
  max_single(5,e);
  max_single(6,f);
  max_single(7,g);
  max_single(8,h);
  delay(5000);
}
```

1.4 Figura

```
#include "figura.h"
#include "comando.h"
#include "max7219.h"

void stop(){

    led8x8(0xc3,0xe7,0x66,0x18,0x18,0x66,0xe7,0xc3);
    led8x8(0xc3,0xe7,0x66,0x18,0x18,0x66,0xe7,0xc3);
    led8x8(0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0);
    led8x8(0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0);
    led8x8(0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0);
    led8x8(0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0);
}
// BITS SETADOS EM CADA LINHA P/ FORMAR A SETA P/ ESQUERDA
/*
  B00010000,
  B00110000,
  B01110000,
  B11111111,
  B11111111,
  B01110000,
  B00110000,
  B00010000
 */
void seta_esquerda(){

    led8x8(0x18,0x3c,0x7e,0xff,0x18,0x18,0x18,0x18);
    led8x8(0x18,0x3c,0x7e,0xff,0x18,0x18,0x18,0x18);
    led8x8(0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0);
    led8x8(0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0);
    led8x8(0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0);
    led8x8(0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0);
}

//BITS SETADOS EM CADA LINHA P/ FORMAR A SETA P/ DIREITA
/*
  B00001000,
  B00001100,
  B00001110,
  B11111111,
```

```
  B11111111,
  B00001110,
  B00001100,
  B00001000
 */

// FUNÇAO SETA P/ DIREITA

void seta_direita(){

      led8x8(0x18,0x3c,0x7e,0xff,0x18,0x18,0x18,0x18); //direita
      led8x8(0x18,0x3c,0x7e,0xff,0x18,0x18,0x18,0x18);
      led8x8(0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0);
      led8x8(0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0);
      led8x8(0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0);
      led8x8(0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0);

    //write8x8(0x18,0x18,0x18,0x18,0xff,0x7e,0x3c,0x18);
}

//FUNÇAO SINALIZAÇAO - TODOS OS LED LIGADOS PARA INDICAR O CICLISTA

void sinalizaçao(){

    led8x8(0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff); //sinalizaçao
    led8x8(0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff);
    led8x8(0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0);
    led8x8(0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0);
    led8x8(0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0);
    led8x8(0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0);

}
```

E. Código para interação entre GSM e GPS na IDE Energia

```
#include <TinyGPS++.h>
TinyGPSPlus gps;
double latitude;
double longitude;
void setup() {
Serial.begin(9600);
}
void loop() {
smartDelay(1000); /* Generate precise delay of 1ms */
if (gps.location.isValid())
{
latitude = gps.location.lat(),8;
longitude = gps.location.lng(),8;
Serial.println(latitude);
Serial.println(longitude);
sendsms();
```

```
}
else{
latitude = 1;
longitude = 1;
sendsms();
}
}
static void smartDelay(unsigned long ms)
{
unsigned long start = millis();
do
{
while (Serial.available()) /* Encode data read from GPS while data
is available on serial port */
gps.encode(Serial.read());
/* Encode basically is used to parse the string received by the GPS
and to store it in a buffer so that information can be extracted from it
*/
} while (millis() - start < ms);
}
void sendsms()
{
Serial.print(" AT+CMGF=1\r");
delay(500);
Serial.print("AT+CMGS =\"+5561995--------\"\r");
delay(500);
Serial.print(latitude);
Serial.print(longitude);
Serial.print("\r");
delay(500);
Serial.print(0x1A);
}
```

F. Código do GSM para receber mensagem no CCS

```
#include "msp430g2553.h"
#include "uart.h" // Arquivar o arquivo uart com o codigo main


void gsm(); // Prototipo de funçao
int main(void){

   WDTCTL = WDTPW + WDTHOLD;// Para o o whatchdogtimer
   BCSCTL1 = CALBC1_8MHZ;// Faz a frequencia de 8Mhz com o launchpad
   DCOCTL = CALDCO_8MHZ;



   uart_init(); // CHAMAR A FUNÇÃO UART INIT QUE ESTÁ DISPONÍVEL NO ARQUIVO
   __enable_interrupt();//  Interrupçao ENABLE
   __delay_cycles(100000);
   gsm();// Chama a funçao gsM
}
```

```
void gsm(){
    uart_puts((char *)"AT"); // Comando para inicializar o GSM
    uart_putc(0x0A);
    uart_putc(0x0D);//retorna carriege
    __delay_cycles(10000000);// ATRASO ... ESPERE OK DE GSM
    uart_puts((char *)"AT+CMGF=1");//Comunicaçao
    uart_putc(0x0A);
    uart_putc(0x0D);
   __delay_cycles(10000000); // espera pelo Ok
    uart_puts((char *)"AT+CMGS=\"+5561995094992\""); //manda a mensagem para esse numero
    uart_putc(0x0A);
    uart_putc(0x0D);
    uart_puts((char *)"LOCALIZAÇAO: ... "); //MANDA A LOCALIZAÇAO PARA O NUMERO ... AINDA FALTA INTREGRAR
    uart_putc(0x1A);// "ctrl Z"


}
```