Centro Federal de Educação Tecnológica de Santa Catarina – CEFET-SC

Curso de Pós-Graduação em Desenvolvimento de Produtos Eletrônicos Digitais

LEITOR MICROCONTROLADO DE GPS – *GLOBAL POSITIONING SYSTEM* (SISTEMA DE POSICIONAMENTO GLOBAL)

ROBERTO DO AMARAL SALES

FLORIANÓPOLIS/SC

2008

Centro Federal de Educação Tecnológica de Santa Catarina – CEFET-SC
Curso de Pós-Graduação em Desenvolvimento de Produtos Eletrônicos Digitais

LEITOR MICROCONTROLADO DE GPS – *GLOBAL POSITIONING SYSTEM* (SISTEMA DE POSICIONAMENTO GLOBAL)

Monografia apresentada ao Curso de Pós-Graduação em Desenvolvimento de Produtos Eletrônicos Digitais como parcial à obtenção do título de Especialista em Desenvolvimento de Produtos eletrônicos Digitais

Orientador:
Prof. Dr. Golberi de Salvador Ferreira

FLORIANÓPOLIS/SC
2008

Roberto do Amaral Sales

Monografia sob o título Leitor Microcontrolado de GPS, defendido por Roberto do Amaral Sales em 30 de maio de 2008 e aprovado pela banca examinadora constituída conforme abaixo:

_____

Prof. Dr. Golberi de Salvador Ferreira

_____

Prof Dr. Jony Laureano Silveira

_____

Prof Dr. Muriel Bittencourt de Liz

FLORIANÓPOLIS/SC

2008

DEDICATÓRIA

Este estudo é dedicado aos inúmeros pesquisadores que tem empreendido seus esforços a tornar o Brasil reconhecido mundialmente como um país desenvolvedor de alta tecnologia.

"Ser educador é ser um poeta do amor. Educar é acreditar na vida e ter esperança no futuro. Educar é semear com sabedoria e colher com paciência" - Augusto Cury - Psicoterapeuta, escritor e cientista.

AGRADECIMENTOS

Primeiramente a Deus, que provê todas as coisas, a minha amada esposa Ana Lúcia, a meus pais, o amigo Fábio Pereira, meus mestres, e por fim a todos que me inspiraram e contribuíram para a realização desta pesquisa, meus sinceros votos de agradecimento.

SUMÁRIO

1. INTRODUÇÃO.....10
2. O AVANÇO DO GEOPOSICIONAMENTO.....11
3. GPS – SISTEMA DE POSICIONAMENTO GLOBAL 13
   3.1. TRILATERAÇÃO.....17
4. MICROCONTROLADORES E GPS 22
5. PADRÃO NMEA 0183 24
   5.1. SENTENÇA RMC 28
   5.2. SENTENÇA GGA.....29
6. MÓDULO GPS OEM SERIAL.....31
7. MICROCONTROLADOR PIC 16F877.....34
8. MICROCONTROLADOR ARM STR711.....39
9. COMUNICAÇÃO COM O RECEPTOR GPS.....42
   9.1. AQUISIÇÃO DA SENTENÇA NMEA 0183.....43
   9.2. VALIDAÇÃO DA SENTENÇA NMEA 0183.....45
   9.3. PRÉ-ANÁLISE DA SENTENÇA.....50
   9.4. EXTRAÇÃO DOS DADOS DA SENTENÇA NMEA 0183 52
   9.5. CÁLCULO DA DISTÂNCIA ENTRE DUAS COORDENADAS 59
10. CONCLUSÃO.....68
11. GLOSSÁRIO.....70
12. REFERÊNCIAS BIBLIOGRÁFICAS 71
13. ANEXOS.....73

ÍNDICE DE FIGURAS

Figura 1: Distribuição dos satélites GPS.....13
Figura 2: Modulação dos sinais GPS.....15
Figura 3: Trilateração 2D 18
Figura 4: Trilateração 3D 19
Figura 5: Trilateração 3D 19
Figura 6: Satélites muito próximos.....20
Figura 7: Boa geometria entre os satélites 21
Tabela 1: Código ASCII padrão 25
Tabela 2: Código ASCII estendido.....26
Figura 8: Receptor GPS.....32
Figura 9: Conector do receptor GPS 32
Figura 9: Kit didático de microcontroladores.....35
Figura 10: Kit didático de microcontroladores.....36
Figura 11: periféricos do kit didático de microcontroladores.....37
Figura 12: Placa controladora. Detalhe para o microcontrolador ARM.....40
Figura 13: Placa controladora montada com display touch screen.....40
Figura 14: fluxograma da aquisição dos dados do receptor GPS.....45
Figura 15: fluxograma da validação da sentença NMEA 0183 49
Figura 16: fluxograma da pré-análise da sentença NMEA 0183.....52
Figura 17: fluxograma da atualização do ponteiro 54
Figura 18: fluxograma da extração dos dados.....58
Figura 19: Distância dos grandes círculos.....61

RESUMO

Desde as mais antigas civilizações, o ser humano sempre teve necessidade de se localizar no mundo. Os mapas rudimentares, as inscrições e demarcações em pedra, os monumentos usados como pontos de referência geográfica, tudo indica a vontade humana em se perceber no espaço e demarcar seu território. Esta monografia procura mostrar o avanço no posicionamento global através do GPS – *Global Positioning System* (Sistema de Posicionamento Global), bem como mostrar o quanto a tecnologia está acessível para projetos com eletrônica embarcada. Propõe-se como prática a elaboração de uma biblioteca em linguagem C, compatível com diversas plataformas, para facilitar a integração de receptores de GPS com microcontroladores e microprocessadores em sistemas onde é necessário criar uma "cerca geográfica", adquirindo dos receptores informação de hora, data, coordenadas geográficas, e fazendo o cálculo da distância entre duas coordenadas distintas. Para a criação do código, será focada a capacidade dos processadores, as diferenças entre cada compilador e as possíveis aplicações desta biblioteca.

1. INTRODUÇÃO

Este trabalho apresenta de forma prática uma biblioteca desenvolvida na linguagem de programação C, para realizar o processo de interpretação dos dados enviados por um receptor GPS. Este dispositivo eletrônico é capaz de fornecer a posição geográfica através da medição de sinais oriundos de uma rede de satélites, disponibilizando os dados da medição em variáveis que podem ser processadas por um sistema eletrônico, com o propósito de obter coordenadas geográficas precisas, atendendo a aplicações que necessitam de tal informação.

A biblioteca foi desenvolvida de tal forma que ela possa ser usada em diversas plataformas, desde um microcontrolador de pequeno porte até um microcomputador, sem a necessidade de adaptação ou qualquer mudança de código. Para atingir o resultado, foram usados comandos comuns a todas as linguagens.

O objetivo primário do projeto é servir de base para sistemas de geoposicionamento facilitando o trabalho do programador, de forma que o processo de programação e uso dos receptores GPS seja transparente, como se o programador usasse uma ferramenta do próprio ambiente de programação.

O objetivo secundário é servir de base para futuros projetos de pesquisa ou trabalhos de conclusão de curso, que utilizem módulos receptores de GPS.

As bibliotecas foram testadas em três plataformas diferentes: um microcontrolador PIC16F877, um microcontrolador ARM STR711 e em um computador utilizando o compilador Borland TURBO C++ 2008.

O trabalho foi organizado em tópicos que abordam os conceitos fundamentais do geoposicionamento, os microcontroladores utilizados para o desenvolvimento da biblioteca, as características da comunicação entre os microcontroladores e os receptores de GPS, e por fim a elaboração das diversas partes de código que compõem a biblioteca.

2. O AVANÇO DO GEOPOSICIONAMENTO

Com o crescente avanço da tecnologia, tornam-se mais simples, viáveis e compactos os equipamentos eletrônicos digitais. Atualmente, diversas tecnologias sofisticadas passam a fazer parte do nosso cotidiano.

Uma das ferramentas é o GPS. Através de um receptor GPS, é possível definir em instantes a nossa localização no globo terrestre, com precisão de até alguns centímetros, dependendo do receptor utilizado.

Devido à miniaturização e fabricação em larga escala, este dispositivo está cada vez mais presente em nosso cotidiano.

É possível ver no comércio de produtos eletrônicos diversos aparelhos com navegador GPS embutidos como, por exemplo, celulares, navegadores pessoais veiculares (em substituição aos tradicionais mapas), rastreadores de veículos, organizadores pessoais, notebooks, e diversos outros equipamentos estão surgindo diariamente.

As aplicações, antes restritas à localização, cartografia, topografia, estão ficando cada vez mais inusitadas, sendo limitadas apenas pela imaginação do desenvolvedor.

Tendo a possibilidade de verificar instantaneamente a posição geográfica, é possível idealizar uma série de dispositivos para nos ajudar no dia a dia.

Imagine poder colocar uma pulseira na mão do seu filho pequeno ou um detalhe na roupa dele, que colete as informações precisas de localização, troque dados com seu telefone e provoque um alarme sonoro se a criança se afastar de você. E mesmo se isso acontecer, pense que você ainda saberá a localização da criança. Seria muito útil em locais muito movimentados, ou com diversos corredores e salas.

Outro exemplo de uso do GPS é o controle patrimonial. Atualmente o sistema já é usado no rastreio de veículos, mas imagine se pudessem ser acoplados em outros itens, como notebooks, celulares, computadores em locais públicos, caixas eletrônicos e outros equipamentos sujeitos a furto ou vandalismo, facilitando o controle patrimonial do bem. Com a miniaturização, em um futuro próximo um rastreador GPS poderia ser incorporado em obras de arte, como adicional no controle patrimonial.

Empresas que locam equipamentos caros poderiam evitar que os mesmos fossem locados para terceiros, na tentativa de minimizar os gastos com o aluguel. Isso provoca o desgaste prematuro das peças da máquina, refletindo em custos de manutenção que obviamente são repassados para o cliente.

Se a empresa que loca os equipamentos tivesse controle do mesmo, muitas manutenções seriam evitadas e o aluguel poderia baixar proporcionalmente. Ou seja, com o uso racional do equipamento, todos saem ganhando.

Para se conseguir esse resultado, bastaria fazer com que o equipamento funcionasse apenas dentro de uma área de habilitação reconhecida pelo receptor GPS, desmotivando a prática comum de usar a máquina além dos limites de operação normal. Essa prática ocorre principalmente com alguns equipamentos eletromédicos.

Existem diversas técnicas para o rastreio e/ou localização.

Algumas técnicas fazem uso dos sistemas de telefonia celular, outras exigem uma infra-estrutura dedicada, com antenas de rádio, mas sem dúvida, o que apresenta a melhor precisão com menor investimento é o GPS.

No próximo capítulo será descrito o sistema de posicionamento global e suas origens, a fim de conhecer melhor o funcionamento do sistema.

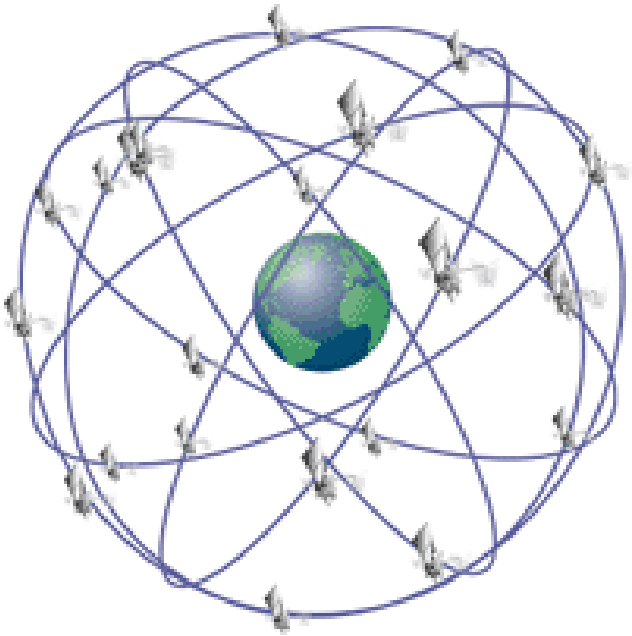
3. GPS – SISTEMA DE POSICIONAMENTO GLOBAL

O sistema de posicionamento global foi criado pelo Departamento de Defesa dos Estados Unidos (DoD) na década de 60, e divulgado como totalmente operacional em meados de 1995.

O nome inicial do projeto era “Projeto NAVSTAR”, cuja sigla significa “Sistema de Navegação com Tempo e Alcance”, do inglês “*Navigation System with Time and Ranging*”. O objetivo primário era somente o uso militar, mas foi liberado o uso civil, com restrições na precisão.

O primeiro satélite foi lançado em 1978, no entanto em 1994 já havia uma constelação de 24 satélites. Cada satélite tem como estimativa uma vida útil de pelo menos 10 anos.

O sistema é composto de 24 satélites principais e mais quatro reservas. Estão divididos em seis órbitas distintas a uma altitude de 20200 km e um plano orbital de 55 graus em relação ao plano equatorial.



Fonte: GPS Global

Figura 1: Distribuição dos satélites GPS

O planejamento das órbitas e distribuições dos satélites foi feito de tal forma que em qualquer ponto do planeta, sempre existam pelo menos quatro satélites visíveis, o que garante a cobertura, como sugere a figura 1. Em função do movimento terrestre e do movimento orbital dos satélites, a quantidade de satélites visíveis muda em um mesmo local, mas sempre em número superior a quatro satélites rastreáveis.

Os satélites demoram 12 horas para completar uma órbita, então cada satélite percorre duas voltas completas diariamente ao redor da terra.

A energia fundamental dos satélites provém de células solares, porém no caso de um eclipse solar, baterias mantém o seu funcionamento.

As órbitas são bastante estáveis, todavia cada satélite é equipado com pequenos propulsores para garantir sua posição e curso durante sua vida útil.

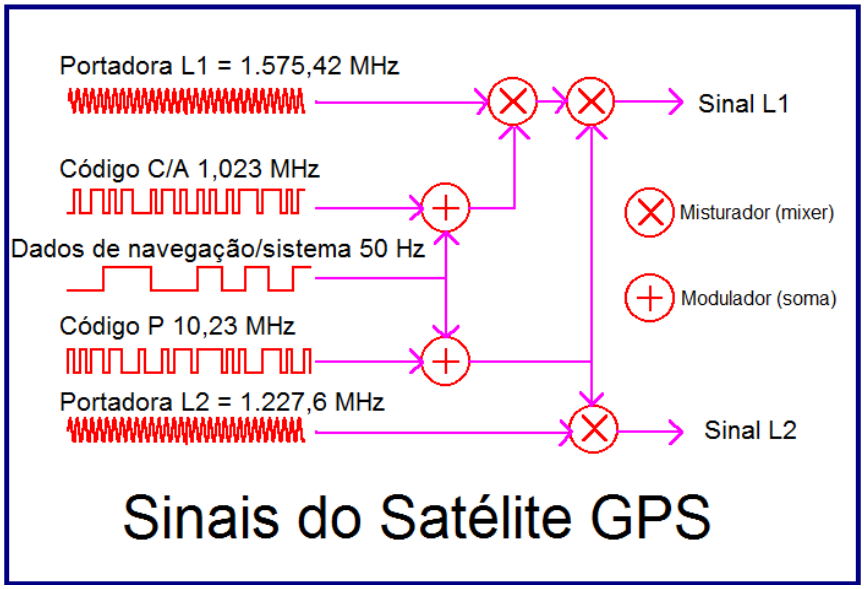
Os satélites enviam basicamente três tipos de sinais:

- O código pseudo-aleatório informa qual satélite está enviando o sinal;
- Os dados de efemérides, do latim *ephemèris ìdis* que significa "memorial diário", "calendário", informam a data, hora e a situação do satélite;
- Os dados de almanaque informam a posição de todos os satélites.

Os militares americanos tem acesso a um tipo de código chamado código "P" – de "Preciso", do inglês *Precise*. O código P é um recurso capaz de realizar medições com altíssima precisão. Para realizar tais medições, são usadas duas freqüências, denominadas L1 e L2. O receptor mede o atraso entre os sinais distintos e calcula o erro, corrigindo os efeitos causados pela atmosfera.

O outro tipo de código chama-se "CA" – "Aquisição Grosseira", do inglês *Coarse Acquisition*. O código CA é gerado apenas na freqüência L1. Portanto, como não existe outra referência, não é possível determinar o erro causado pela atmosfera. Sendo assim, a precisão é afetada pela difração atmosférica e reflexão em possíveis obstáculos. O código CA é a base dos sistemas civis.

Conforme mostra a figura 2, os sinais de navegação do GPS tem uma taxa de transmissão de 50 bits por segundo. Estes bits são codificados usando dois padrões distintos: o código CA e o código P. A freqüência base para todos os processos de modulação e codificação do sinal são gerados a partir de um clock de 10,23 MHz.



Fonte: própria

Figura 2: Modulação dos sinais GPS

O código CA é gerado a partir do divisor por 10 da freqüência base de 10,23 MHz, que resulta em 1,023 MHz. O código P é gerado diretamente a partir da freqüência base de 10,23 MHz.

Os códigos CA e P têm características de um ruído aleatório, mas são precisamente definidos. Por esse motivo tem a designação *Pseudo Randon Code* – PRN, ou código pseudo-aleatório.

Os sinais de navegação são então somados em módulo 2 aos códigos CA e P. Dessa forma, obtemos um sinal resultante codificado. Cada bit do código CA ou P recebe o nome de "*chip*", pois não contém nenhuma informação. Dadas as características da geração dos códigos, um chip CA se repete a cada milissegundo e um chip P se repete a cada uma semana.

O fato de repetir um bit do código somente a cada uma semana torna o código P extremamente complexo, de forma que não existam receptores capazes de decodificar o código P.

Esta técnica é chamada *anti spoofing*, que significa "anti interceptação".

Os códigos gerados a partir do CA e P são então modulados em duas freqüências distintas.

O código gerado a partir de CA é modulado apenas na freqüência portadora L1, que é igual a 150 vezes a freqüência base de 10,23 MHz, ou seja, 1575,42 MHz.

O código gerado a partir de P é modulado tanto na freqüência L1, como também na freqüência portadora L2, que é igual a 120 vezes a freqüência base de 10,23 MHz, ou seja, 1227,6 MHz. O código P é modulado nas duas freqüências para que o atraso entre as duas freqüências seja base de cálculo para determinar os erros causados pela atmosfera, tornando o código P mais preciso.

Um fator que tornava a medição imprecisa era o sincronismo entre os satélites. Propositalmente, os sinais eram gerados atrasados ou adiantados, sendo que o receptor desconhecia tal informação. Se o sinal não chegava no tempo certo, o cálculo era feito de forma errônea que gerava uma posição desviada da posição real. Esta técnica se chamava “disponibilidade seletiva”, ou *Selective Availability*.

O governo americano desabilitou o sistema em maio de 2000, aumentando significativamente a precisão dos equipamentos de GPS, que antes tinham um erro de 30 a 100 metros, passando a uma precisão média de 15 metros. Atualmente está ativo apenas o “bloqueio regional”, ou *Regional Deniability*, que produz uma degradação proposital em áreas onde o governo americano possui determinados interesses militares.

Os satélites têm um relógio extremamente preciso, sendo usados não somente para localização, mas também para fins científicos e técnicos onde a precisão é um item fundamental. Esse tipo de relógio que equipa os satélites é um relógio atômico, com precisão de nanossegundos. O que torna o sistema GPS a referência de tempo mais estável e de menor custo usada mundialmente, uma vez que os receptores de GPS ao redor do mundo recebem a mesma informação de hora, com precisão de microssegundos.

A potência de transmissão de um sinal GPS é de apenas 50 W, sendo que ao chegar ao receptor em terra, o sinal é muito fraco. Filtros extremamente precisos separam o sinal do ruído de fundo da própria terra, para posterior processamento.

O sinal, depois de filtrado, é enviado para um circuito eletrônico. Cada circuito eletrônico é reconhecido como sendo um canal do receptor GPS. Se o receptor tem apenas um canal, ele pode processar os dados de um dos satélites de cada vez, tornando o processo lento. Embora o processo não seja rápido, em menos de um segundo já existe uma trilateração pronta disponibilizada pelo receptor GPS.

Os receptores que possuem mais canais podem processar mais satélites simultaneamente, possibilitando o uso em maiores velocidades de deslocamento, devido o erro causado pela mudança da posição do medidor durante o cálculo. Em

velocidades elevadas o cálculo se torna impossível, pois ao término do processamento de um sinal de satélite, o mesmo já está desatualizado, implicando em erros e imprecisão.

Para obter a posição de um objeto na superfície terrestre, o sistema GPS funciona segundo um princípio simples da física: se sabemos a velocidade e o tempo do percurso, podemos calcular a distância.

Este processo é a trilateração. A trilateração é a forma com que os receptores de GPS calculam a posição de um ponto na superfície terrestre.

3.1. TRILATERAÇÃO

Os satélites enviam os códigos simultaneamente e o receptor calcula o tempo que levou para o código chegar até ele. Assim, calcula-se a distância do receptor até o satélite. Sabendo a distância de pelo menos três satélites, podem-se calcular as coordenadas geográficas de um ponto da superfície terrestre e sabendo-se a distância de quatro satélites, é possível ainda calcular a altitude do referido ponto.

Para explicar a trilateração, vamos tomar um exemplo em duas dimensões.

Se você está perdido em uma região e alguém te diz:

– Você está a 370 km de Santos.

Logo você percebe que poderia estar em qualquer lugar na grande circunferência a 370 km de Santos.

Se outra pessoa lhe diz:

– Você também está a 480 km de Cascavel.

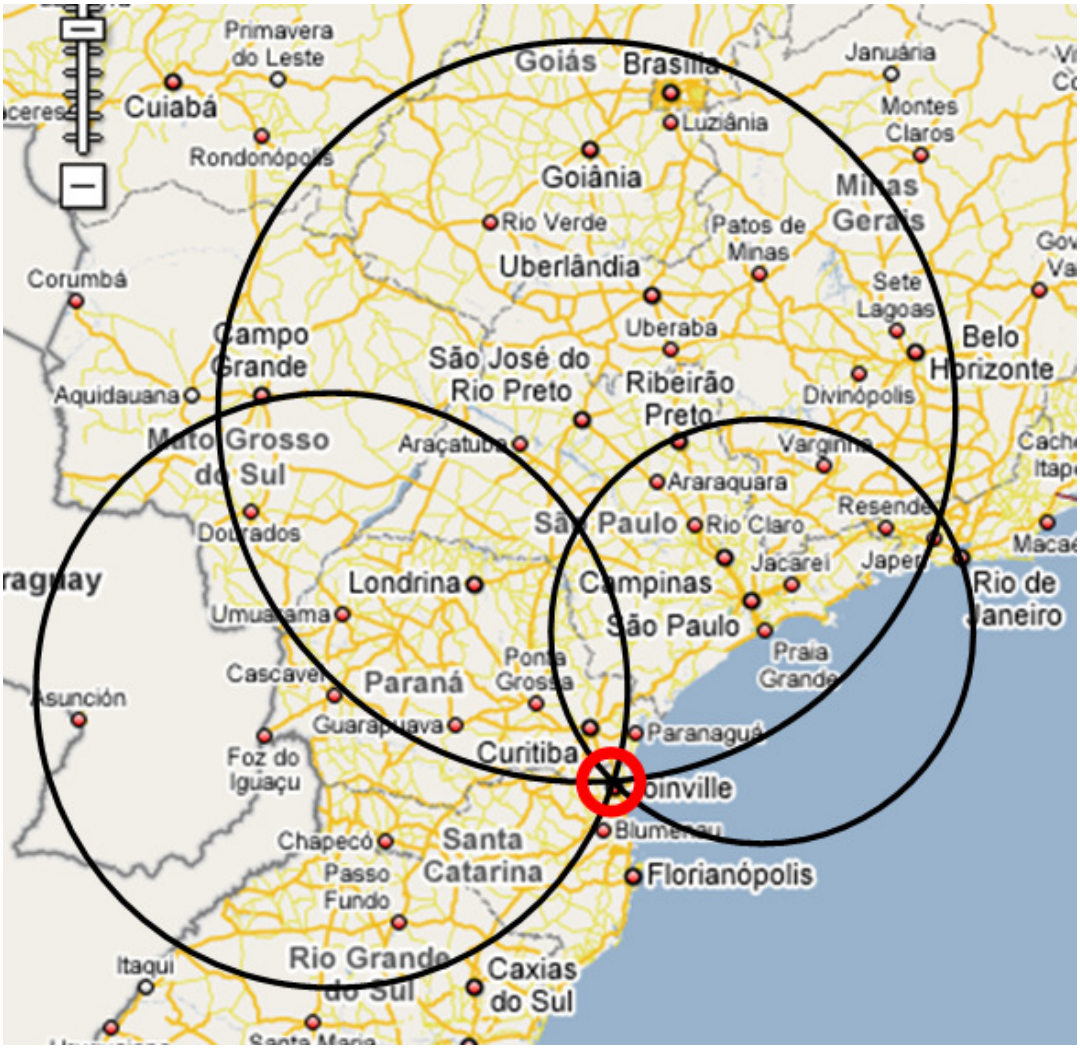
Restam-lhe duas opções, pois as duas grandes circunferências se cruzam em dois pontos distintos.

Se uma terceira pessoa lhe disser:

– Você está a 610 km de São José do Rio Preto.

Em duas dimensões, foram necessárias três circunferências para obter uma localização geográfica.

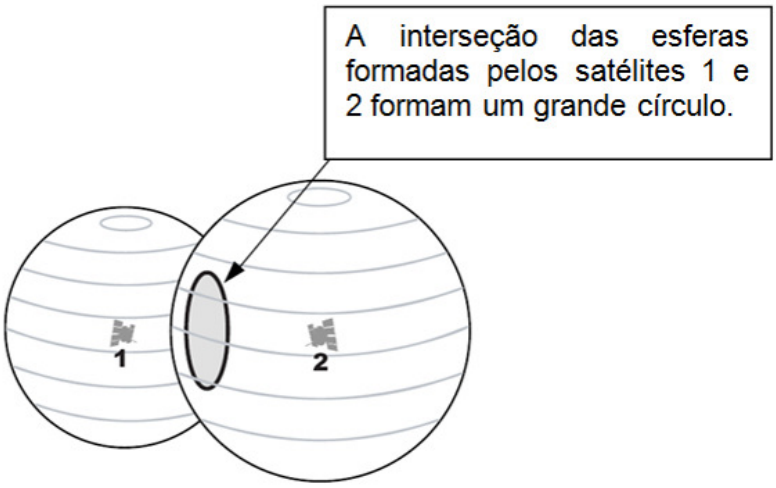
Você pode afirmar com certeza que está em Joinville, pois apenas em Joinville ocorre o cruzamento destas três grandes circunferências, como mostra a Figura 3.



Fonte: própria

Figura 3: Trilateração 2D

Em três dimensões, o processo ocorre da mesma forma, com uma sutil diferença que é o fato de termos inicialmente não um círculo, mas uma esfera. Desta forma, teremos sempre uma dimensão a mais. Veja na figura 4 como isso ocorre.



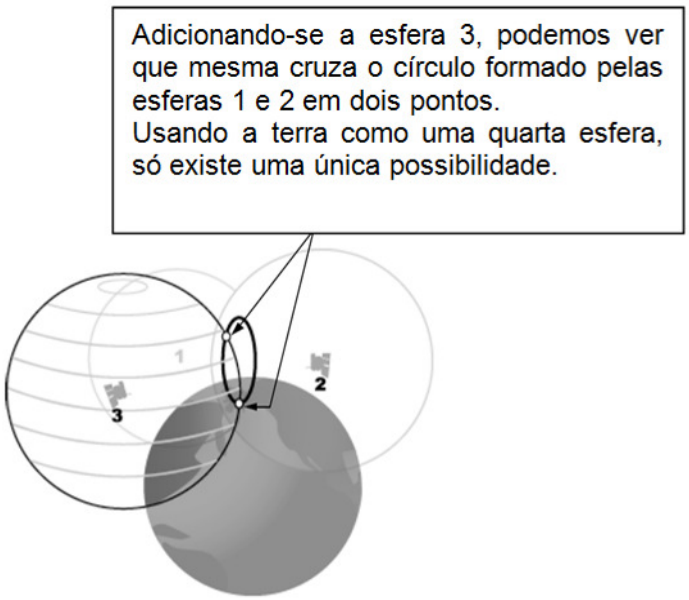
Fonte: Garmin

Figura 4: Trilateração 3D

Com dois satélites, teremos na interseção das esferas um grande círculo.

Somando-se o sinal de um terceiro satélite, teremos dois pontos de encontro.

Usando-se a terra como a quarta esfera, teremos o ponto único de encontro de todos os sinais. Levando-se em consideração que a terra não é uniforme, podemos afirmar que a medição de altura não é precisa (figura 5).



Fonte: Garmin

Figura 5: Trilateração 3D

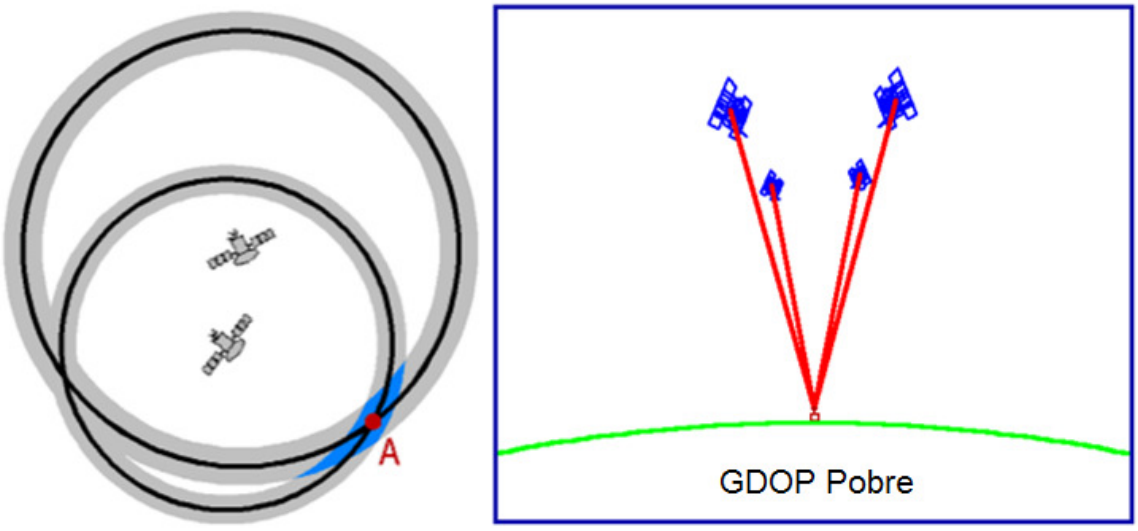
Mas se tivermos um quarto satélite ao invés da terra, podemos medir a altitude de um objeto com a mesma precisão que medimos as coordenadas.

Para realizar os cálculos, o receptor GPS usa o relógio interno, que não é tão preciso quanto os relógios atômicos que equipam os satélites. Por isso, a medição adquire um erro.

Os sinais enviados pelo GPS podem sofrer desvios na ionosfera, provocando outro erro. Como os sinais podem percorrer multipercursos, os mesmos acabam por chegar defasados ao receptor, provocando o aumento da distância medida pelo mesmo.

Os sinais ainda podem ser refletidos por obstáculos e isso provoca mais um erro. Um prédio muito alto ou uma montanha ou qualquer outro obstáculo de grandes dimensões pode refletir ou bloquear o sinal do satélite, aumentando as possibilidades de medição errônea.

Caso os satélites estejam muito próximos entre si (figura 6), ocorre um erro devido a essa geometria entre eles. As zonas de imprecisão se sobrepõem de tal forma que elas aumentam muito o tamanho.

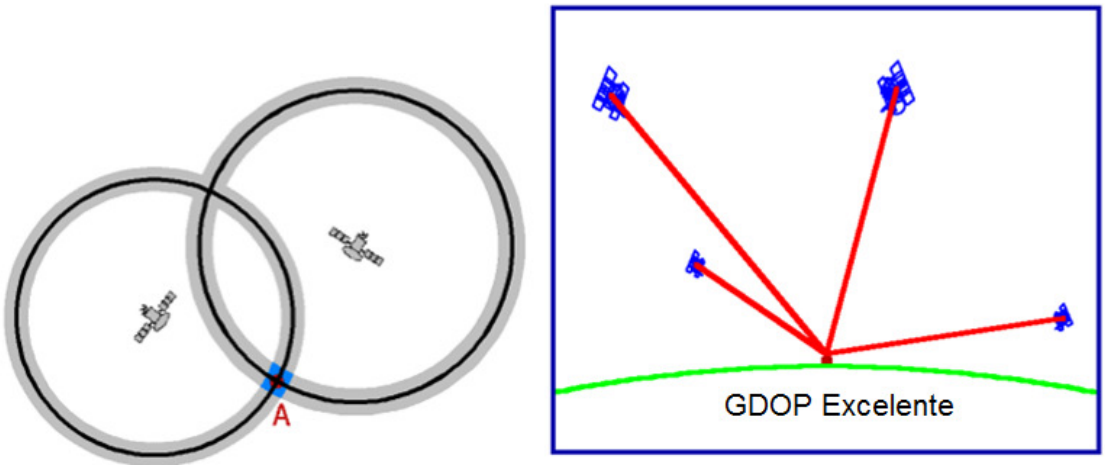


Fonte: kowoma

Figura 6: Satélites muito próximos

Se os satélites estiverem em uma boa posição (figura 7) as zonas de imprecisão se cruzam em uma área bem pequena, aumentando a precisão da medida.

O aumento da distância entre os satélites aumenta a consistência da medição, mas pode trazer outro problema. Como os satélites estarão em ângulos críticos, um obstáculo pode obstruir sua visada, como por exemplo, quando a pessoa está junto a prédios ou em meio à vegetação cerrada ou cadeia de montanhas, prejudicando a chegada do sinal. O ideal é que os satélites não estejam nem muito próximos e nem muito afastados para que o ângulo com o receptor não seja nem muito fechado nem muito aberto, mas todos os satélites possam ser enxergados sem problemas.



Fonte: kowoma

Figura 7: Boa geometria entre os satélites

Essa geometria entre os satélites é informada através do parâmetro “diluição da precisão” do inglês *Dilution of Precision* ou *Geometric Dilution of Precision* (DOP ou GDOP), que quanto menor o seu valor, melhor a medição.

Ao final do processo de medição, são somados diversos erros, causando imprecisão das medidas. Mesmo assim, deve-se considerar que não existe nenhum sistema que se compare ao GPS em precisão.

Após todo o processo de aquisição dos sinais e cálculo das coordenadas, o receptor de GPS disponibiliza estas informações na interface de dados, para o processamento por um sistema eletrônico, normalmente microcontrolado. Acontece então a integração entre os microcontroladores e o GPS. No próximo capítulo será abordada a forma com que os microcontroladores e receptores de GPS podem ser integrados em um mesmo projeto.

4. MICROCONTROLADORES E GPS

Muitos módulos GPS são comercializados em regime OEM – *Original Equipment Manufacturer*, ou fabricante de equipamento original.

Um equipamento OEM é um dispositivo pronto, que vai ser integrado em um produto final. O dispositivo é fabricado por um terceiro, por isso a denominação fabricante de equipamento original.

No caso do GPS, várias empresas especializadas fabricam módulos, que podem ser integrados em um produto, simplificando e barateando a fabricação do mesmo.

Apesar da oferta de módulos, o problema gerado é: “como fazer o módulo funcionar com a minha aplicação?”.

Existem muitos módulos no mercado, mas os aplicativos ainda não têm bibliotecas prontas para os mesmos.

Muitas pessoas desistem do uso por desconhecer totalmente o funcionamento do GPS. Como as interfaces são padronizadas e o protocolo de dados é simples, as aplicações com GPS são muito fáceis de serem embutidas em qualquer plataforma.

Para facilitar o uso dos receptores de GPS, é proposto como resultado deste trabalho a elaboração de uma biblioteca para a decodificação das sentenças mais importantes e extração dos dados da mesma.

A biblioteca pode ser usada como base para futuros trabalhos acadêmicos, bem como para projetos comerciais, sendo a mesma completamente funcional para obter data, hora, coordenadas geográficas e distância entre duas coordenadas.

Para criar uma biblioteca modular, foram adotados alguns procedimentos descritos a seguir, tornando a mesma expansível em recursos.

Para garantir a precisão e confiabilidade das medidas, foram adotados recursos para verificar a validade da sentença recebida pelo receptor GPS.

Para a elaboração da biblioteca para GPS, é necessário seguir corretamente o padrão adotado pelos fabricantes destes equipamentos para os dados disponibilizados na interface de comunicação.

Mundialmente é adotado o padrão NMEA 0183, que define não somente o formato dos dados, mas também a interface elétrica utilizada.

Para conseguir cumprir os objetivos deste trabalho, foi proposto o seguinte:

- Entender como funciona o GPS;
- Estudo do padrão usado pelos receptores GPS (NMEA 0183);
- Estudo das plataformas usadas;
- Criar uma rotina em C para armazenar uma sentença NMEA 0183 para posterior processamento;
- Criar uma rotina em C para validar a sentença NMEA 0183;
- Criar uma rotina em C para extrair os dados de latitude, longitude, data e hora de uma sentença NMEA 0183;
- Converter os dados para o formato correto para posterior processamento;
- Criar uma rotina para cálculo de distância, a partir de duas coordenadas;
- Validar todas as rotinas nas plataformas utilizadas.

Veremos a seguir como foi idealizado este padrão e como ele é empregado nos sistemas de navegação, bem como suas características fundamentais.

5. PADRÃO NMEA 0183

A Associação Nacional de Eletrônica Naval dos Estados Unidos (*National Marine Electronics Association* ou NMEA) desenvolveu uma especificação técnica que define a interface entre os diferentes equipamentos eletrônicos marítimos. A norma permite aos equipamentos eletrônicos o envio de informações para computadores e outros equipamentos marítimos.

O padrão NMEA 0183 é usado pelos instrumentos de navegação marítima para enviar dados aos equipamentos onde estão conectados. O mesmo define a interface elétrica e o protocolo de dados utilizado nessa conexão. A comunicação do receptor GPS está definida no âmbito desta especificação.

Originalmente, a interface de dados recomendada é a EIA422, mas é aceito para muitas aplicações a interface RS-232.

Todas as sentenças NMEA 0183 são do tipo texto ASCII.

ASCII é o acrônimo para *American Standard Code for Information Interchange*, que em português significa "Código Padrão Americano para o Intercâmbio de Informações". O código ASCII foi inventado em 1961 para que os sistemas computacionais funcionassem de forma padronizada, tendo um código em binário para representar cada caractere ou símbolo de pontuação, símbolo gráfico ou número.

Dessa forma, os sistemas computacionais poderiam trocar informações entre si, fato impossível antes de 1961, pois cada caractere era representado de uma forma diferente nas diversas máquinas da época.

Os caracteres que podem ser impressos (letras, números, pontuação, etc.) são numerados de 32 a 127, enquanto que os caracteres de 0 a 31 são caracteres de controle dos sistemas computacionais, como as teclas *tab* (9), *enter* (13) e a tecla *esc* (27).

Com a difusão do código ASCII, viu-se a necessidade de expandir a tabela para acomodar códigos de outras línguas e alguns outros símbolos utilizados nos programas computacionais e foi criado o código ASCII estendido, compreendido entre o número 128 e 255.

O código ASCII se tornou padrão mundial e é usado em praticamente todos os sistemas que envolvem algum tipo de processamento, desde microcontroladores até computadores de grande porte. O código ASCII é muito usado nos sistemas de comunicações, desde uma simples porta serial até comunicação através da internet.

A tabela 1 mostra o código ASCII padrão, de 0 a 127 em decimal, hexadecimal e o caractere correspondente, enquanto que na tabela 2, vemos a tabela ASCII estendida, representada da mesma forma.

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

Fonte: Cégep de Drummondville

Tabela 1: Código ASCII padrão

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
128	80	Ç	160	A0	á	192	C0	└	224	E0	α
129	81	ü	161	A1	í	193	C1	┴	225	E1	ß
130	82	é	162	A2	ó	194	C2	┬	226	E2	Γ
131	83	â	163	A3	ú	195	C3	├	227	E3	π
132	84	ä	164	A4	ñ	196	C4	─	228	E4	∑
133	85	à	165	A5	Ñ	197	C5	┼	229	E5	σ
134	86	å	166	A6	ª	198	C6	╞	230	E6	µ
135	87	ç	167	A7	º	199	C7	╟	231	E7	τ
136	88	ê	168	A8	¿	200	C8	╚	232	E8	Φ
137	89	ë	169	A9	⌐	201	C9	╔	233	E9	Θ
138	8A	è	170	AA	¬	202	CA	╩	234	EA	Ω
139	8B	ï	171	AB	½	203	CB	╦	235	EB	δ
140	8C	î	172	AC	¼	204	CC	╠	236	EC	∞
141	8D	ì	173	AD	¡	205	CD	═	237	ED	ø
142	8E	Ä	174	AE	«	206	CE	╬	238	EE	ε
143	8F	Å	175	AF	»	207	CF	╧	239	EF	∩
144	90	É	176	B0	░	208	D0	╨	240	F0	≡
145	91	æ	177	B1	▒	209	D1	╤	241	F1	±
146	92	Æ	178	B2	▓	210	D2	╥	242	F2	≥
147	93	ô	179	B3	│	211	D3	╙	243	F3	≤
148	94	ö	180	B4	┤	212	D4	╘	244	F4	⌠
149	95	ò	181	B5	╡	213	D5	╒	245	F5	⌡
150	96	û	182	B6	╢	214	D6	╓	246	F6	÷
151	97	ù	183	B7	╖	215	D7	╫	247	F7	≈
152	98	ÿ	184	B8	╕	216	D8	╪	248	F8	°
153	99	Ö	185	B9	╣	217	D9	┘	249	F9	∙
154	9A	Ü	186	BA	║	218	DA	┌	250	FA	·
155	9B	¢	187	BB	╗	219	DB	█	251	FB	√
156	9C	£	188	BC	╝	220	DC	▄	252	FC	ⁿ
157	9D	¥	189	BD	╜	221	DD	▌	253	FD	²
158	9E	₧	190	BE	╛	222	DE	▐	254	FE	■
159	9F	ƒ	191	BF	┐	223	DF	▀	255	FF	

Fonte: Cégep de Drummondville

Tabela 2: Código ASCII estendido

As sentenças começam com o caractere “$” e terminam com um retorno de carro e alimentação de linha (<CR>+<LF>, que são respectivamente os códigos 13 e 10 da tabela ASCII). Os dados são separados por vírgula, sendo as mesmas obrigatórias por serem os delimitadores de dados. Nem todos os campos estão disponíveis em todos os equipamentos, embora as vírgulas estejam presentes para demarcar os diferentes campos de dados.

Isto sugere que as sentenças NMEA 0183 têm largura variável, pois um ou mais campos de dados podem ser omitidos, tornando impossível fixar a posição de um dado na sentença.

Um campo de checksum é adicionado opcionalmente ao final da sentença.

O checksum é uma forma de proteger a integridade das informações, através da detecção de erros nos dados que foram enviados ou armazenados. O campo de checksum é concatenado ao final de uma transmissão de dados ou ao final de um arquivo armazenado. Quando o sistema recebe estes dados ou lê, ele refaz os cálculos e compara os dois valores. Se forem iguais, o sistema assume que os dados são verdadeiros.

Conforme [WAKERLY], o checksum funciona bem para grandes sentenças de dados, pois a probabilidade de combinações aumenta significativamente. Segundo o mesmo autor, o checksum apresenta como desvantagens a necessidade de ler todo o código para constatar um provável erro e a impossibilidade de encontrar a origem do erro.

Analisando a estrutura da sentença NMEA 0183, seguindo o caractere “$” os cinco primeiros caracteres formam o campo de endereço, onde os dois primeiros caracteres são “GP” (apenas no nosso caso) para identificar que é uma sentença de GPS e os três seguintes são o tipo de sentença NMEA 0183, que serve para formatar o conteúdo dos dados transmitidos. Sabendo o tipo de sentença NMEA 0183, podemos construir um “parser”, que é um programa dedicado à interpretação da sentença e extração dos dados.

Existem diversas sentenças NMEA 0183, atendendo as mais diversas aplicações, sendo que as mais usadas são as listadas abaixo.

$ GPAAM – Alarme de chegada à rota

$ GPALM – Dados de almanaque GPS (Pode também ser recebido pela unidade GPS)

$ GPAPB – Piloto automático formato "B"

$ GPBOD – Rolamento, da origem para o destino

$ GPBWC – Rolamento e distância para uma rota, grande círculo

$ GPGGA – Sistema de Posicionamento Global, dados corrigidos

$ GPGLL – Posição geográfica, latitude / longitude

$ GPGRS – GPS residuais da escala

$ GPGSA – GPS DOP e satélites ativos

$ GPGST – GPS Estatísticas do ruído da pseudoescala

$ GPGSV – Satélites GPS em linha de visada

$ GPHDT – Verdadeira posição do navio em graus

$ GPMSK – Controle de um receptor Beacon

$ GPMSS – Status de um receptor Beacon

$ GPR00 – Lista de pontos do percurso na rota atualmente ativa

$ GPRMA – Mínimo recomendado para dados específicos Loran-C

$ GPRMB – Mínimo recomendado para informações de navegação

$ GPRMC – Mínimo recomendado para dados específico GPS / TRÂNSITO

$ GPRTE – Rotas

$ GPTRF – TRÂNSITO Dados corrigidos

$ GPSTN – Dados múltiplas identidades

$ GPVBW – Velocidade Terra / Água

$ GPVTG – Direção do percurso em graus e velocidade

$ GPWPL – Localização da rota

$ GPXTE – Erro do trajeto, medido

$ GPZDA – Data / Hora UTC e correção para fuso horário local

Das sentenças listadas, as mais importantes para o trabalho aqui desenvolvido e também as sentenças enviadas pelo receptor adotado no projeto são a RMC e GGA, cujo formato se explica a seguir.

5.1. SENTENÇA RMC (Mínimo recomendado para dados específico GPS / TRÂNSITO)

O formato de uma sentença RMC é o que segue:

$GPRMC,	hhmmss.ss,	A,	llll.ll,	a,	yyyyy.yy,	a,	x.x,	x.x,	ddmmyy,	x.x,	a*hh
	1	2	3	4	5	6	7	8	9	10	11

Onde,

O campo 1 traz hora, minuto, segundo e centésimos de segundo, de acordo com o UTC.

O campo 2 traz informações da medição: A=ativo, V=alerta.

O campo 3 traz latitude (2 primeiros dígitos) em graus, minutos (próximos 3 dígitos) e fração de minutos (últimos 2 dígitos).

O campo 4 traz a orientação da latitude: N=norte, S=sul.

O campo 5 traz a longitude (3 primeiros dígitos) em graus, minutos (próximos 3 dígitos) e fração de minutos (últimos 2 dígitos).

O campo 6 traz a orientação da longitude: E=leste, W=oeste.

O campo 7 traz a velocidade sobre o solo em nós.

O campo 8 traz o curso real.

O campo 9 traz o dia, mês e ano, todos com dois dígitos e nessa ordem.

O campo 10 traz a variação magnética.

O campo 11 traz a orientação da variação magnética.

Após o símbolo “*” existe o checksum, com dois dígitos.

Exemplo de uma sentença RMC:

$GPRMC,003245.000,V,2618.2160,S,04851.2583,W,0.00,331.25,201207,,,E*7E

5.2. SENTENÇA GGA (Sistema de Posicionamento Global, dados corrigidos)

O formato de uma sentença GGA é o que segue:

$GPGGA,	hhmmss.ss,	llll.ll,	a,	yyyyy.yy,	a,	x,	xx,	x.x,	x.x,	M,	x.x,	M,	x.x,	xxxx*	hh
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Onde,

O campo 1 traz hora, minuto, segundo e centésimos de segundo, de acordo com o UTC.

O campo 2 traz latitude (2 primeiros dígitos) em graus, minutos (próximos 3 dígitos) e fração de minutos (últimos 2 dígitos).

O campo 3 traz a orientação da latitude: N=norte, S=sul.

O campo 4 traz a longitude (3 primeiros dígitos) em graus, minutos (próximos 3 dígitos) e fração de minutos (últimos 2 dígitos).

O campo 5 traz a orientação da longitude: E=leste, W=oeste.

O campo 6 traz a qualidade da medição: I=invalido, Medição GPS, Medição GPS diferencial.

O campo 7 traz a quantidade de satélites sendo rastreados.

O campo 8 traz a diluição horizontal da precisão, HDOP sendo que quanto menor o valor, mais preciso é a medida.

O campo 9 traz a altitude em metros, acima do nível do mar.

O campo 10 traz a grandeza da medição da altitude.

O campo 11 traz a altura do nível do mar acima da elipsóide de referência WGS84.

O campo 12 traz a grandeza da medição da altitude.

O campo 13 traz o tempo em segundos desde a última atualização DGPS.

O campo 14 traz o número da estação DGPS.

Exemplo de uma sentença GGA:

$GPGGA,003246.000,2618.2160,S,04851.2583,W,6,00,50.0,107.5,M,0.9,M,,0000*54

Os módulos GPS encontrados no mercado seguem o especificado no padrão NMEA 0183. Estes módulos podem ter recursos de navegação e localização, ou apenas prover os dados de localização, sem nenhuma interface com o usuário. Para poder ser integrado como parte de um projeto eletrônico, não se torna obrigatório nenhuma interface com o usuário na maioria dos casos.

Será apresentado no próximo capítulo um módulo GPS OEM serial, para uso em sistemas embarcados.

6. MÓDULO GPS OEM SERIAL

Vamos estudar um tipo de receptor GPS comercial para ser integrado em uma aplicação eletrônica.

Existem inúmeros módulos receptores de GPS no mercado. Alguns módulos são destinados a aplicação em sistemas embarcados ou produtos eletrônicos.

Estes módulos podem vir encapsulados ou com o circuito a mostra.

A interface dos módulos pode ser USB, serial, paralela, dependendo da aplicação.

A função do módulo é receber os sinais dos satélites GPS e processar as informações, retornando em sua interface as sentenças NMEA 0183, que serão analisadas e processadas pelo equipamento ao qual elas foram ligadas.

A velocidade da interface de dados é 4800 bps, como definido no padrão NMEA 0183, sendo o formato da transmissão 8 bits de dados, sem paridade, um bit de parada ao final do dado.

As dimensões reduzidas do módulo receptor GPS tornam possível embutir o mesmo em diversos sistemas embarcados. O consumo de energia não favorece o uso de baterias, mas o módulo pode ser desligado em momentos onde não é interessante o monitoramento da posição geográfica.

Quando energizados, a transmissão dos dados é automática e periódica. O módulo envia uma sentença NMEA 0183 a cada 2 segundos, aproximadamente. Neste intervalo, ocorre o processamento das informações pelo sistema embarcado.

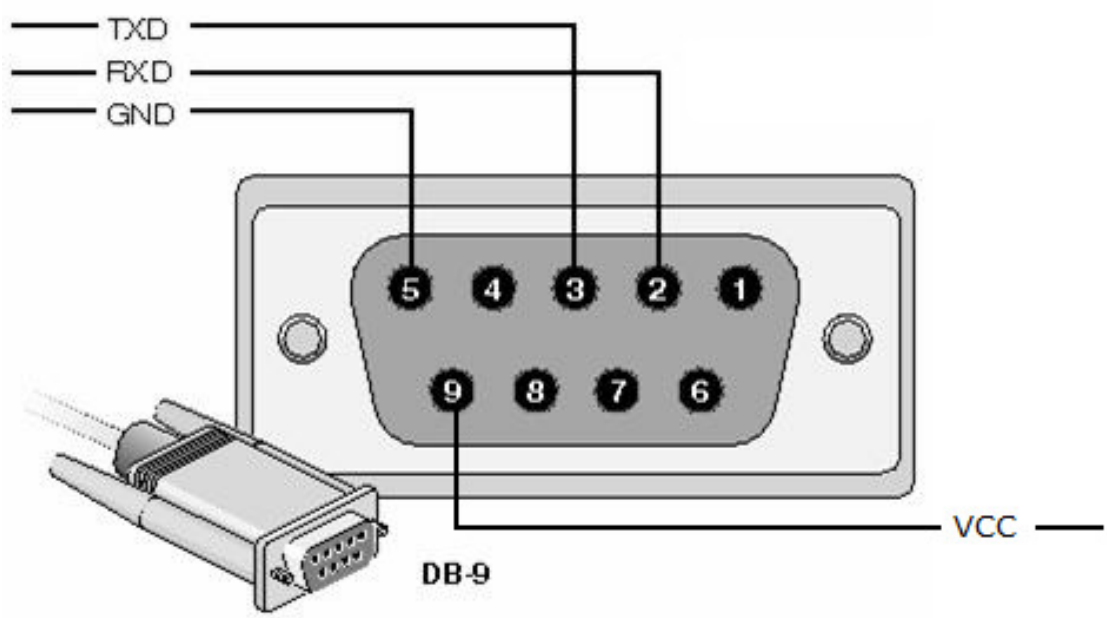
Neste projeto foi utilizado um módulo receptor GPS com chipset SiRFIII, equipado com 12 canais (figura 8).

O módulo conta com uma interface elétrica RS-232, compatível com conector de 9 pinos DB-9.

A configuração dos pinos do conector é mostrada na figura 9. O receptor possui um pino de transmissão de dados, um pino de recepção de dados, um pino de terra de referência e alimentação e um pino de alimentação positiva, que deve ser ligado a uma fonte de tensão estabilizada de 5 volts contínuos.

Fonte: própria

Figura 8: Receptor GPS



Fonte: própria

Figura 9: Conector do receptor GPS

Nota-se que o receptor de GPS não possui nenhuma funcionalidade, se não estiver ligado a um sistema que faça o processamento das informações. Para um sistema eletrônico simples, podemos usar um microcontrolador de 8 bits, pois a quantidade de informações a ser processada é pequena.

Será apresentado a seguir um modelo de microcontrolador de 8 bits, escolhido para esta aplicação por ser fácil de encontrar no comércio de componentes eletrônicos e também pelo fato de ser amplamente utilizado no meio acadêmico.

7. MICROCONTROLADOR PIC 16F877

Os microcontroladores PIC da série 16 são microcontroladores de 8 bits de baixa performance, com recursos limitados, típicos de aplicações mais simples, onde não é exigido alto poder de processamento de dados.

Estes microcontroladores são fabricados pela empresa Microchip e são populares mundialmente.

O microcontrolador usado no projeto é um PIC16F877. Ele possui 40 pinos, 8 KB de memória de programas, 368 bytes de memória RAM, porta serial USART, SPI E I2C, porta paralela escrava, portas analógicas e comparadores analógicos.

Atualmente, este é o microcontrolador mais popular neste segmento e é encontrado facilmente no mercado. Este foi o principal critério de escolha deste microcontrolador.

O oscilador é externo e pode ser utilizado até 20 Mhz, que produz uma velocidade de execução de 5 milhões de instruções por segundo.

O microcontrolador não possui funções matemáticas mais complexas implementadas no código, sendo que o mesmo executa apenas soma, subtração, deslocamento para a direita e esquerda (divisão e multiplicação por 2).

Com estas características, vemos que o microcontrolador é adequado para controle, envolvendo operações simples. Assim sendo, temos de nos preocupar com a complexidade do código, de forma que não hajam atrasos devido ao tempo de execução das instruções.

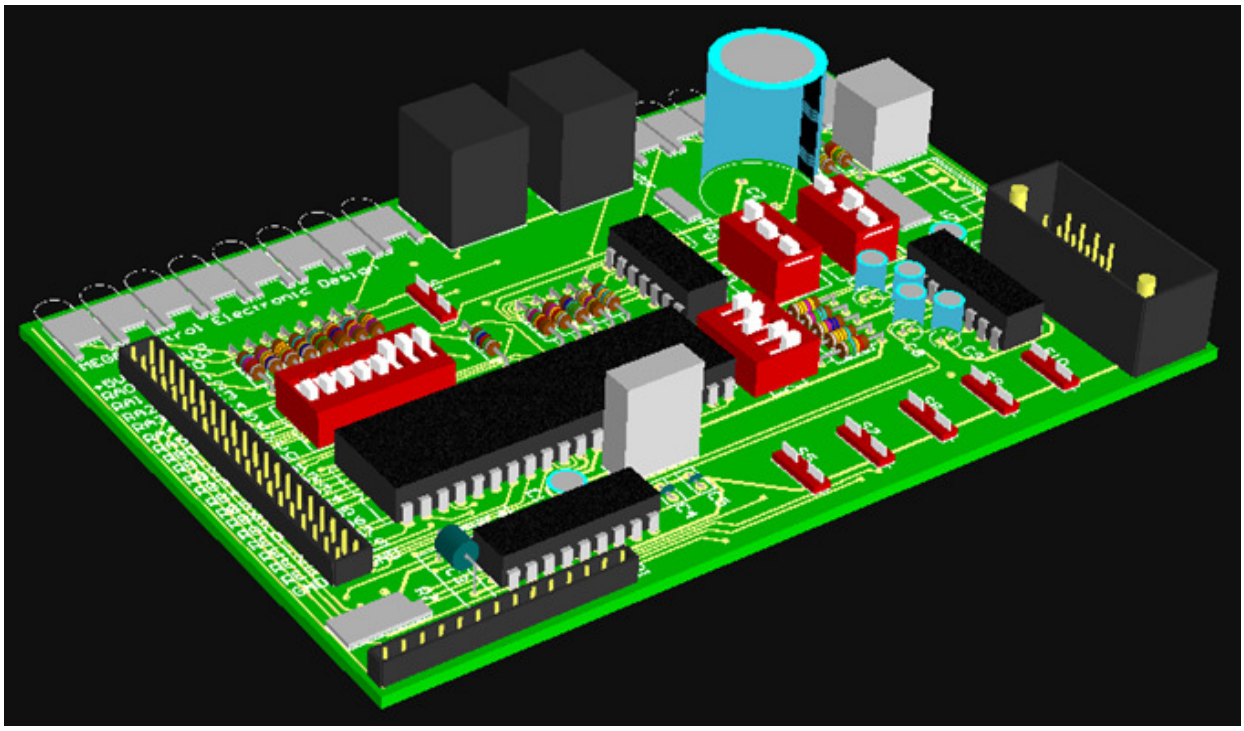
O ambiente de desenvolvimento escolhido foi o PCWH, fabricado pela empresa *Custom Computer Services Inc* – conhecida como CCS.

O software permite a programação de uma vasta gama de microcontroladores da empresa *Microchip*, cobrindo todos os modelos comerciais nas famílias 12, 16 e 18, que são os microcontroladores de pequeno e médio porte fabricados pela *Microchip*.

Para os testes, foi usado um kit didático (figura 10) desenvolvido pelo autor para uso na SOCIESC, onde o mesmo leciona. O kit conta com cristal de 4 MHz, conversor de nível para comunicação serial e módulo LCD.

O kit didático foi construído para possibilitar a programação de microcontroladores PIC fabricados pela empresa Microchip, com opções de uso de diversos modelos, inclusive com tamanhos físicos diferentes, possibilitando seu uso em nível profissional, para incentivar o aluno a continuar o aperfeiçoamento após a conclusão das disciplinas.

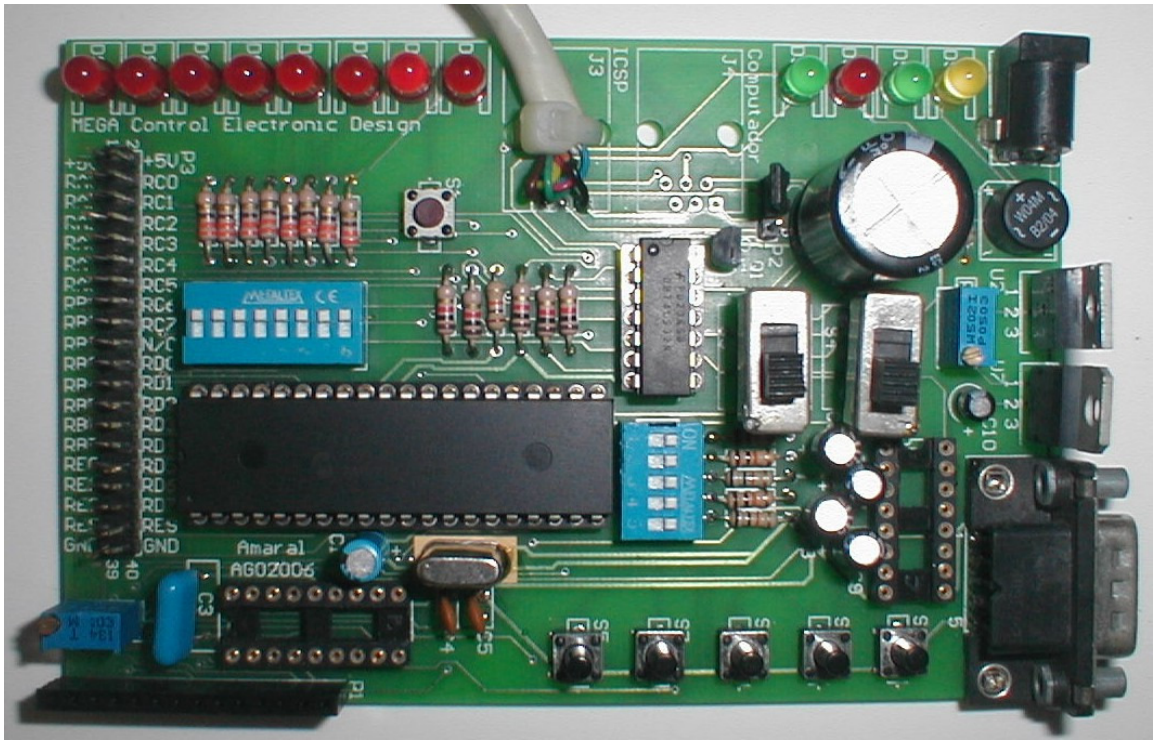
A placa é construída com componentes de baixo custo, de fácil aquisição no mercado local, para que o mesmo possa ser de propriedade do aluno. Desta forma, ele pode praticar em casa, criar suas próprias experiências, ter a disciplina como hobby.



Fonte: própria

Figura 10: Kit didático de microcontroladores

Adotando-se uma metodologia lúdica, o aluno passa a sentir a necessidade de descobrir o universo da programação de microcontroladores, se divertindo com as coisas que ele pode fazer em casa, com a experiência de poder ter controle sob equipamentos e poder automatizar muitas tarefas em casa. Enfim, o aluno passa a construir o conhecimento em cima da sua própria necessidade pessoal. Isso é a força propulsora chamada motivação.



Fonte: própria

Figura 11: Kit didático de microcontroladores

O valor do kit didático é cerca de 20% do valor de um kit comercial, podendo custar até menos de 12% do valor de um kit semelhante caso seja construído em larga escala (acima de 500 unidades por mês). O kit foi idealizado para estar ao alcance da maior quantidade de pessoas difundindo o conhecimento em todas as classes sociais.

O kit é bastante compacto e não exige ferramentas especiais para a montagem, podendo ser confeccionado pelo próprio aluno.

Para facilitar o uso, existem leds indicadores de funcionamento na placa, além de leds conectados aos pinos do microcontrolador para simular experiências, botões com a mesma finalidade, porta serial para conectar outros equipamentos, expansão para conectar periféricos, expansão para conectar displays de cristal líquido. O kit finalizado pode ser visto na figura 11.

Mesmo sem os periféricos o aluno pode realizar uma série de experiências, pois uma grande parte das experiências para principiantes pode ser feita utilizando-se leds e botões para verificar o funcionamento.

Fonte: própria

Figura 12: periféricos do kit didático de microcontroladores

Diversos periféricos foram desenvolvidos para que os alunos possam aproximar as aulas do seu cotidiano de uma forma lúdica. Fazem parte da lista de periféricos equipamentos novos, desenvolvidos em parte pelos alunos e equipamentos reciclados. Exemplos de periféricos estão ilustrados na figura 12.

Para visualizar os eventos e fornecer uma interface para a programação dos mesmos, é utilizado um display de cristal líquido inteligente, que é facilmente encontrado em sucatas de equipamentos eletrônicos e lojas de componentes eletrônicos.

O conector serial do receptor de GPS conecta-se diretamente à placa do kit didático e o display utilizado para verificar o funcionamento das bibliotecas e para realizar as experiências também se conecta diretamente à placa, em outro conector específico para esta finalidade.

Caso o programador deva desenvolver uma aplicação mais complexa, será necessário um maior poder de processamento.

Alguns microcontroladores possuem configurações que permitem desenvolver aplicações muito avançadas, contando com instruções complexas, mais memória de programas, mais memória de dados e outros recursos que contribuem para desenvolver aplicações de alto nível.

No próximo capítulo será apresentada uma tecnologia e um dos microcontroladores mais usados mundialmente: o microcontrolador ARM.

8. MICROCONTROLADOR ARM STR711

Ao contrário do que muitos pensam ARM não é um chip e sim uma arquitetura. Hoje, a arquitetura ARM pertence à empresa *Advanced Risc Machines*, e os microcontroladores ARM são fabricados a partir de acordos comerciais sob o regime de licença. Existem diversos tipos de licença disponíveis. Cada fabricante adquire os direitos que melhor lhe convier.

Os microcontroladores ARM possuem alto desempenho e baixo consumo de energia. É considerado um marco na indústria de semicondutores pelo fato de ser uma arquitetura maciçamente fabricada e rapidamente difundida.

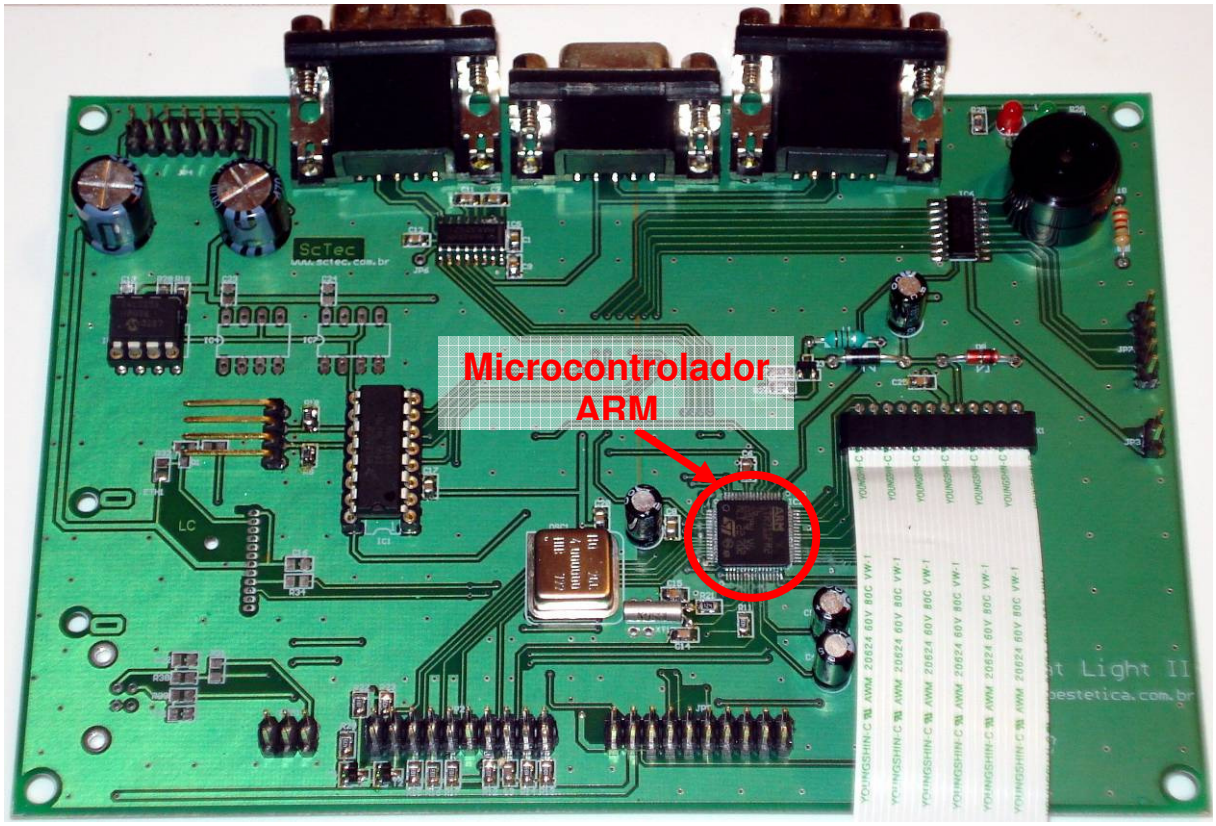
Existem inúmeros modelos de microcontroladores com tecnologia ARM, fabricados por diversas empresas. As ferramentas de software disponíveis para ARM são muitas e uma boa parte gratuitas.

Os microcontroladores ARM estão presentes na maioria dos equipamentos portáteis da atualidade, incluindo telefones celulares, computadores de mão, MP3 players, jogos eletrônicos, etc.

No projeto foi utilizado um microcontrolador ARM STR711, fabricado pela empresa *STMicroelectronics*.

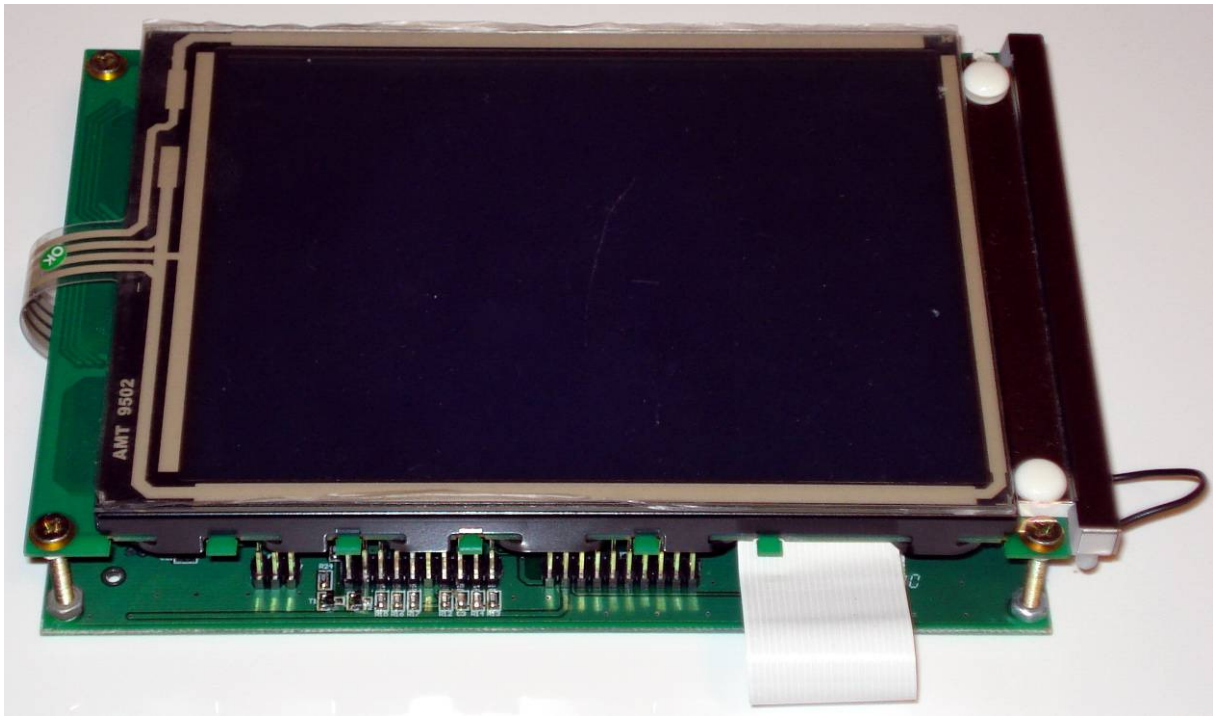
O Ambiente de desenvolvimento utilizado foi o *Embedded Workbench for ARM*, fabricado pela empresa *IAR Systems*.

A placa utilizada no projeto é uma placa controladora padrão desenvolvido pela SCTEC para atender a projetos que demandem de muito processamento e uma interface amigável com o usuário. A placa conta com um display gráfico de 320x240 pixels com *touch screen*. A controladora pode ser observada na figura 13 e a mesma com o display acoplado na figura 14.



Fonte: própria

Figura 13: Placa controladora. Detalhe para o microcontrolador ARM



Fonte: própria

Figura 14: Placa controladora montada com display touch screen

Com as duas plataformas apresentadas até o momento, o projetista pode atender a uma vasta gama de aplicações, sem prejudicar a portabilidade.

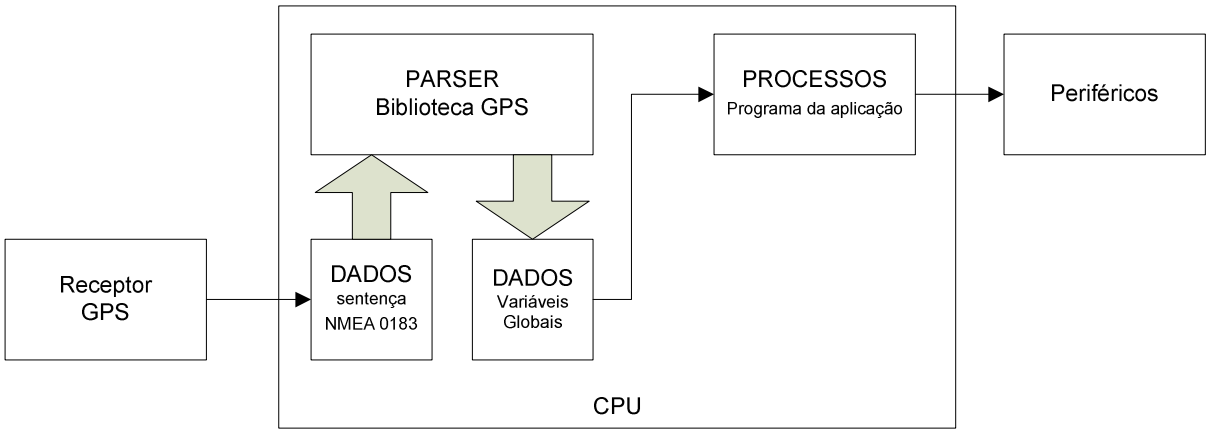
Escolhida a plataforma, deve-se elaborar uma rotina de aquisição dos dados compatível com tal plataforma.

A comunicação com o GPS é um ponto chave na integração do projeto e é sugerido um método para realizar a aquisição dos dados.

9. COMUNICAÇÃO COM O RECEPTOR GPS

Tipicamente, as aplicações devem funcionar segundo o diagrama em blocos da figura 15. Um receptor de GPS será ligado a uma CPU, que pode ser um microcontrolador ou um computador.

O "*parser*", que em linguagem de programação significa "analisador sintático", fará a extração dos dados da sentença NMEA 0183 para que o sistema possa utilizar estas informações em algum programa, a fim de comandar algum processo ou acionar periféricos ligados ao sistema eletrônico, como displays, sirenes, atuadores, etc.



Fonte: própria

Figura 15: diagrama em blocos de um sistema eletrônico com GPS

Para fazer o processamento das informações recebidas do receptor GPS, o primeiro passo foi armazenar e decodificar as informações das sentenças NMEA 0183 recebidas através da porta serial da aplicação.

Para armazenar a sentença, foi elaborada uma rotina de análise para verificar a existência do caractere "$", que marca o início das sentenças NMEA. Quando do recebimento deste caractere, tem início o armazenamento dos caracteres em uma variável do tipo string com 84 caracteres de dimensão. Esse é o número máximo de caracteres de uma sentença NMEA. A variável armazena temporariamente a sentença NMEA a fim de formar uma sentença completa, que será analisada e

posteriormente copiada a outra variável caso ela seja válida, liberando a variável atual para o armazenamento da próxima sentença.

A rotina armazena os caracteres até o recebimento de um retorno de carro <CR> ou até receber 84 caracteres.

Como as plataformas são muito diferentes quanto ao funcionamento das portas seriais, foi elaborado um roteiro para a aquisição da sentença NMEA 0183, descrito a seguir.

9.1. AQUISIÇÃO DA SENTENÇA NMEA 0183

A fim de coletar os dados transmitidos pelo GPS, foi elaborada uma rotina específica, que muda para cada plataforma.

Porém, as ações são as mesmas em todos os sistemas. Devem-se receber os dados e armazená-los em uma variável. Neste instante, os dados ainda não têm um formato utilizável, pois são apenas caracteres. Somente após separar os dados da sentença NMEA é que os mesmos podem ser convertidos, cada um para seu formato correto. Depois que a sentença estiver disponível para o processamento, a mesma é filtrada e os dados são armazenados em outras variáveis em seus respectivos formatos de uso.

Um pequeno obstáculo na aquisição e tratamento dos dados é o fato da sentença NMEA ter largura variável, impossibilitando prever em que posição está o dado. Para possibilitar a análise dos dados, vírgulas são usadas para delimitar os campos.

Como o uso dos delimitadores é obrigatório, nossa análise será na posição de um delimitador dentro da sentença. Se o caractere imediatamente após o delimitador for outro delimitador, sabemos que o campo está vazio. Caso contrário, basta adquirir os dados a partir do delimitador, sabendo que a quantidade de caracteres em um campo é sempre fixa.

Uma preocupação do programador deve ser a validade dos dados da sentença NMEA. Em testes práticos, verificou-se que podem ocorrer perdas de dados devido a ruídos ou atrasos no processamento das informações. Caso haja a perda de um caractere dentro de uma sentença NMEA, a mesma pode até ser completamente comprometida.

Para fazer a validação dos dados, recorre-se ao checksum, enviado ao final da sentença NMEA.

Validando-se a sentença, a mesma pode ser copiada para outra variável a fim de possibilitar o processamento da sentença e extração dos dados.

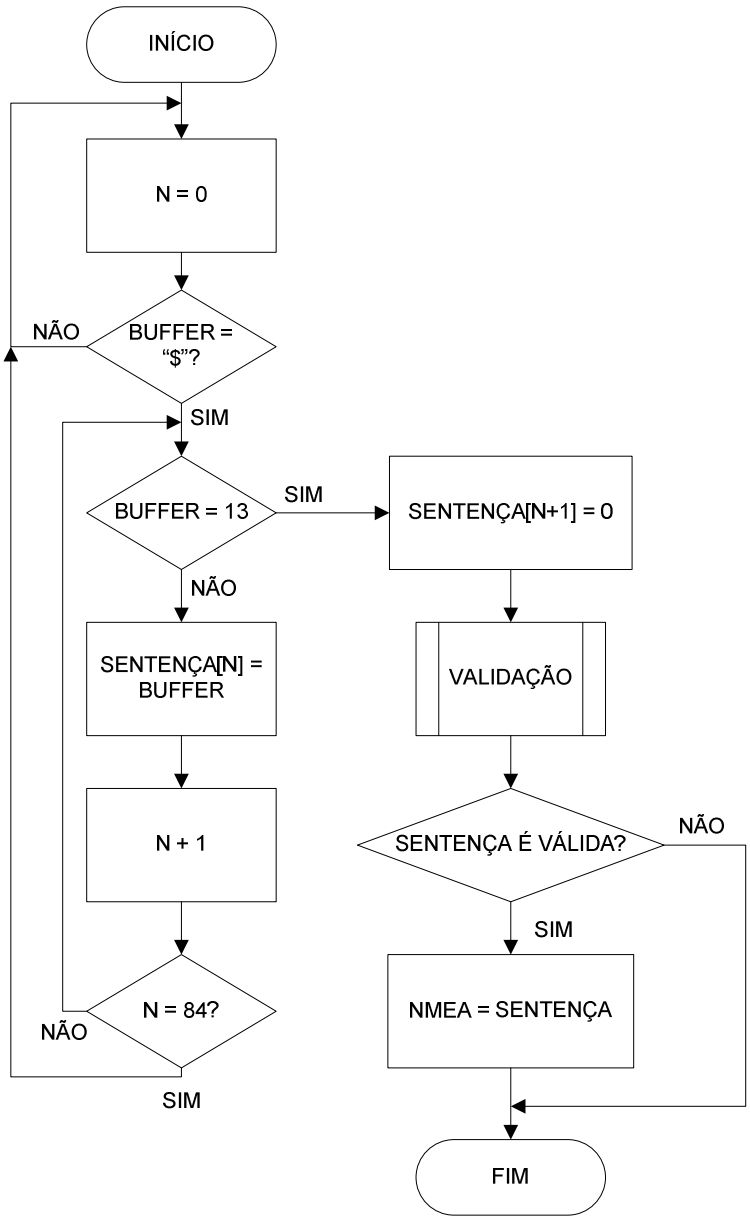
Como estamos trabalhando com dois tipos de sentença NMEA, a RMC e a GGA, o código deve interpretar a sentença e automaticamente se ajustar para poder coletar os dados.

O código analisa os cinco primeiros caracteres da sentença NMEA: se for GPRMC, o código assume uma determinada posição para os dados. Se for GPGGA, é assumida outra posição para os dados. Se não for nenhuma das duas sentenças, nenhum processamento é realizado.

Após o ajuste para a sentença correta, a sentença passa por um contador que retorna a posição do delimitador correto, em função do tipo de sentença NMEA e do campo escolhido.

Sabendo-se a posição do caractere inicial de um campo, basta copiar os caracteres do campo, segundo a quantidade de caracteres e o formato do campo. Durante o processo de cópia, é realizada a conversão dos dados para um formato numérico ou outro qualquer, se necessário.

O fluxograma desta parte do projeto pode ser observado na figura 16.



Fonte: própria

Figura 16: fluxograma da aquisição dos dados do receptor GPS

Após a aquisição dos dados, é necessário validar a sentença NMEA 0183 a fim de garantir a veracidade das informações recebidas. Para tal foi desenvolvido uma biblioteca para analisar a sentença e validar a mesma.

9.2. VALIDAÇÃO DA SENTENÇA NMEA 0183

Para validar a sentença NMEA, é feito uma lógica ou-exclusivo com cada caractere que estiver entre o sinal de "$" e "*", comparando o valor com os dois caracteres que estão após o sinal "*".

Caso os dois números coincidam, a sentença é copiada para uma segunda variável, onde ela poderá ser utilizada normalmente pelo programa, caso contrário, a sentença é sobrescrita no próximo processo de captura da sentença NMEA.

O fluxograma da validação da sentença NMEA 0183 pode ser observado na figura 17, onde o código fonte ficará assim:

```
1     char string_valida()
2     {
3           int checksum,string,a=0;
4           char checksum_string[3];
5           do{
6           a++;  //soma 1 em a até encontrar * ou nulo
7           }while (NMEA[a]!='*'&&a<=83&&NMEA[a]!=NULL);
8                 if (NMEA[a]=='*') {
9                 //se o caractere apontado pela
10                //variável a for *, zera a variável checksum
11                memset(checksum_string,0,3);
12                //repassa os valores do checksum para a variável
13                //checksum_string, para posterior conversão
14                checksum_string[0]=NMEA[a+1];
15                checksum_string[1]=NMEA[a+2];
16                //converte a variável checksum_string de ASCII
17                //para inteiro, gravando em checksum
18                checksum=strtol(checksum_string,0,16);
19                a=1;  // ou a=0;
20                //Se o primeiro caractere é o $, use a=1,
21                //mas se o primeiro caractere for da sentença,
22                //use a=0 para começar a lógica ou-exclusivo
23                string=(NMEA[a]);
24                while (NMEA[a+1]!='*'){
25                a++;
26                //enquanto não encontrar o *, faça lógica
27                //ou-exclusivo do valor atual com o próximo
28                //caractere.
29                string=(string^NMEA[a]);
30                }
31                //se os valores coincidirem,
32                //retorne verdadeiro
33                if (string==checksum) {
34                return true;
```

```
35                  }
36                  else
37                  return false;
38          }
39          else
40          return false;
41    }
```

O código para executar a validação dos dados funciona da seguinte forma:

Inicialmente, são declaradas as variáveis do tipo inteiras “checksum”, “string” e “a” (linha 3).

A variável “a” é inicializada com o valor zero e serve para apontar para o caractere que será analisado, visto que a sentença é analisada caractere a caractere.

A variável checksum guarda o valor recebido como checksum da sentença NMEA 0183, que como visto anteriormente vem logo em seguida ao caractere “*”.

A variável string guarda o valor do checksum calculado conforme o padrão NMEA 0183, que é a lógica ou-exclusivo entre todos os caracteres da sentença compreendidos entre o caractere “$” e o caractere “*”.

É declarado uma variável do tipo matriz de caracteres com o nome “checksum_string” e com dimensão igual a 3 bytes, para armazenar os dois caracteres de checksum e um caractere nulo para terminar a cadeia de caracteres (linha 4).

É criado um laço “do... while”, a fim de encontrar a posição do caractere “*” dentro da sentença NMEA 0183 (linhas 5 a 7). Caso o caractere seja diferente de “*”, a posição menor que 83 e o mesmo diferente de nulo, é somado uma unidade à variável “a”. Ao final, a variável “a” poderá conter a posição do caractere “*”.

Caso o caractere apontado pela variável “a” seja realmente um “*” (linha 8), começa a análise da sentença (linhas 9 a 38), caso contrário (linha 39) a rotina retorna a condição “falso”, indicando que a sentença não corresponde ao que foi enviado pelo receptor de GPS (linha 40).

Da linha 9 até a linha 38 é feito o cálculo do checksum e sua comparação com o valor recebido do receptor de GPS.

A variável checksum_string é preenchida com zeros, apagando seu conteúdo anterior (linha 11).

O caractere após o “*” (na posição a+1) é copiado para a posição zero da variável checksum_string. O próximo caractere (na posição a+2) é copiado para a posição um da variável checksum_string. A última posição é preenchida com um nulo para indicar o término da variável. (linhas 14 e 15).

O valor do checksum está agora na variável checksum_string, porém na forma de texto. O mesmo é convertido para um valor numérico e copiado para a variável checksum (linha 18). Para fazer a conversão é necessário que o terminador da cadeia de caracteres seja nulo. Por isso foi declarada a variável checksum_string com dimensão de 3 bytes e o último caractere foi preenchido com nulo.

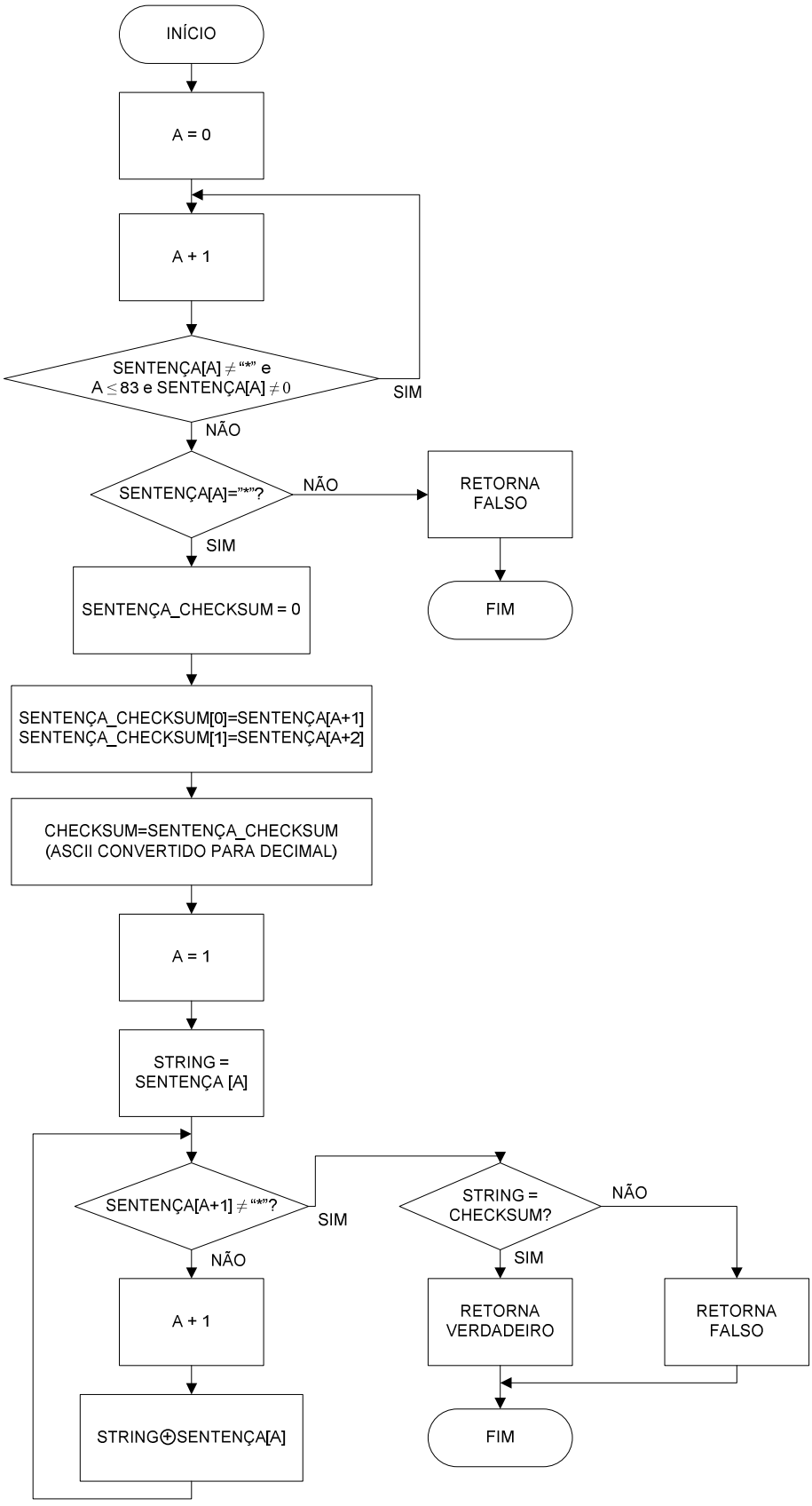
Para iniciar a lógica ou-exclusivo com a sentença NMEA 0183, deve-se verificar a forma como a mesma foi adquirida pelo sistema eletrônico. Caso a sentença comece com o caractere “$”, o mesmo deve ser desprezado, portanto deve-se começar com a variável “a” valendo 1. Caso o caractere “$” já tenha sido removido da sentença, o valor inicial de “a” deve ser 0 (linha 19).

O valor inicial do cálculo do checksum é dado pelo primeiro caractere da sentença (linha 23).

Este valor é calculado através da lógica ou-exclusivo, caractere por caractere até que haja a ocorrência de um caractere “*” na próxima posição (posição a+1). Para isto foi utilizado um laço “while” (linhas 24 a 30).

Enquanto o próximo caractere é diferente de “*”, é feito a lógica ou-exclusivo entre o valor atual do checksum e o valor do caractere, incrementando a posição do caractere na sentença (linhas 25 e 29).

Ao final do cálculo, se o valor da variável checksum for igual ao valor do checksum calculado (linha 33), a rotina retorna a condição “verdadeira” (linha 34), caso contrário (linha 36), retorna a condição “falsa” (linha 37).



Fonte: própria

Figura 17: fluxograma da validação da sentença NMEA 0183

Após a validação dos dados, a sentença está pronta para ser analisada a fim de determinar a posição dos dados dentro da mesma. Deve-se determinar o tipo de sentença, pois a sentença GGA possui os dados em posições diferentes, se comparada a uma sentença RMC. É necessário fazer a pré-análise da sentença e determinar a posição dos ponteiros.

9.3. PRÉ-ANÁLISE DA SENTENÇA

Para que os dados possam ser extraídos de uma sentença NMEA 0183, é fundamental saber a posição destes dados dentro da sentença. Como as sentenças NMEA 0183 têm largura de dados variável, temos que primeiramente determinar a posição dos campos através da análise do tipo de sentença NMEA 0183 e os separadores de campo, que são os caracteres “,”.

O primeiro passo é determinar qual sentença NMEA 1083 estamos trabalhando. Para isso, temos que verificar os primeiros 5 caracteres da sentença NMEA 0183, que informam o tipo de sentença que estamos trabalhando.

No nosso caso, a rotina procura pelos caracteres GGA e RMC. Ocorrendo alguma destas combinações, é definida uma posição para os dados da sentença NMEA 0183, em relação aos separadores de campo.

A posição dos separadores de campo será então usada para extrair os dados da sentença. É chamada uma rotina específica para extrair os dados da sentença.

Caso o sistema não encontre uma sentença NMEA 0183 válida, não ocorre um processamento da mesma, visto que a chamada pela atualização dos dados parte de dentro da rotina de pré análise da sentença.

O fluxograma da pré-análise da sentença NMEA 0183 pode ser observado na figura 18, onde o código fonte ficará assim:

```
1     void atualiza()
2     {
3           if (string_valida()) {
4           //se a sentença for validada,
5           //procura por uma sentença RMC ou GGA.
6           //caso não encontre uma sentença RMC, ou GGA ou
7           //a sentença não seja válida, ocorre o retorno sem
8           //nenhum processamento
```

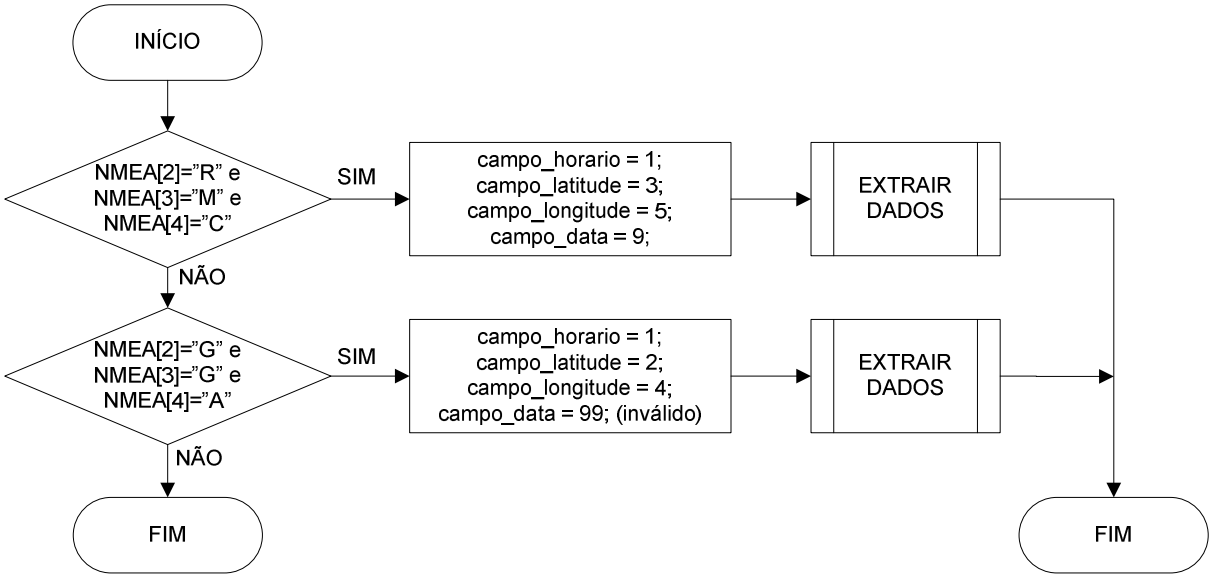
```
9                   if (NMEA[2]=='R'&&NMEA[3]=='M'&&NMEA[4]=='C') {
10                  //campo_horario = 1
11                  //campo_latitude = 3
12                  //campo_longitude = 5
13                  //campo_data = 9
14                  extrai_dados(1,3,5,9);
15                  }
16                  if (NMEA[2]=='G'&&NMEA[3]=='G'&&NMEA[4]=='A') {
17                  //campo_horario = 1
18                  //campo_latitude = 2
19                  //campo_longitude = 4
20                  //campo_data = 99 (inválido)
21                  extrai_dados(1,2,4,99);
22                  }
23          }
24    }
```

O código para executar a pré-análise da sentença NMEA 0183 funciona da seguinte forma:

Se a sentença for válida (linha 3), ocorre a busca pelos códigos GGA e RMC no campo de endereço da sentença NMEA 0183. Caso não seja encontrado o código correto, nenhum processamento é realizado.

Se o código do campo de endereço for “RMC” (linha 9), executa-se uma rotina para a extração dos dados, fixando a posição dos dados conforme o padrão NMEA 0183, ou seja, o campo horário encontra-se após o primeiro separador de campo, o campo da latitude após o terceiro separador de campo, o campo da longitude após o quinto separador de campo e o campo da data após o nono separador de campo (linha 14).

Se o código do campo de endereço for “GGA” (linha 16), a rotina de extração dos dados é executada com outra configuração, conforme o padrão NMEA 0183. O campo horário encontra-se após o primeiro separador de campo, o campo da latitude após o segundo separador de campo, o campo da longitude após o quarto separador de campo e o campo da data, que não está presente na sentença GGA é configurado como 99, para que na rotina de extração dos dados o mesmo seja ignorado.



Fonte: própria

Figura 18: fluxograma da pré-análise da sentença NMEA 0183

Feita a pré-análise dos dados, procede-se a extração dos mesmos, atualizando as variáveis que armazenam estas informações. Mas para cada campo de dados, deve-se primeiramente atualizar a posição dos ponteiros.

9.4. EXTRAÇÃO DOS DADOS DA SENTENÇA NMEA 0183

Após a atualização da posição dos campos de dados da sentença NMEA 0183 em função do tipo de sentença, o sistema automaticamente processa a sentença a fim de extrair os dados.

Para tanto, é necessário em um primeiro momento atualizar o ponteiro de dados para o primeiro caractere que define o campo de dados escolhido e a partir deste coletar os dados seqüencialmente até o final do campo, que tem largura fixa.

A atualização dos ponteiros ocorre da seguinte forma: através do número associado ao campo, podemos contar os caracteres “,” da sentença até ocorrer o devido número, sendo que o próximo caractere na seqüência é o primeiro dado do campo escolhido.

O fluxograma da atualização dos ponteiros pode ser observado na figura 19, onde o código fonte ficará assim:

```
1     void atualiza_ponteiro(int campo_selecionado)
```

```
2     {
3           int a,campo=0;      //começa em campo = 0
4           for (a=0; a <= 83; a++) { //conta de 0 a 83
5                 if (NMEA[a]==',') { //se um dos caracteres é vírgula,
6                 campo++;            //soma um na variável campo
7                 }
8                 if (campo==campo_selecionado) {
9                 //se a variável campo coincide com
10                //a variável escolhida, o próximo caractere
11                //é o ponteiro para o dado a ser extraído e é
12                //terminada a contagem, retornando para a função
13                //que chamou esta
14                ponteiro=a+1;
15                a=83;
16                }
17          }
18    }
```

O código para executar a atualização do ponteiro funciona da seguinte forma:

Duas variáveis do tipo inteiro são inicializadas: “a” e “campo”, sendo que campo é inicializado com o valor zero (linha 3).

A variável “a” aponta para o caractere que será analisado.

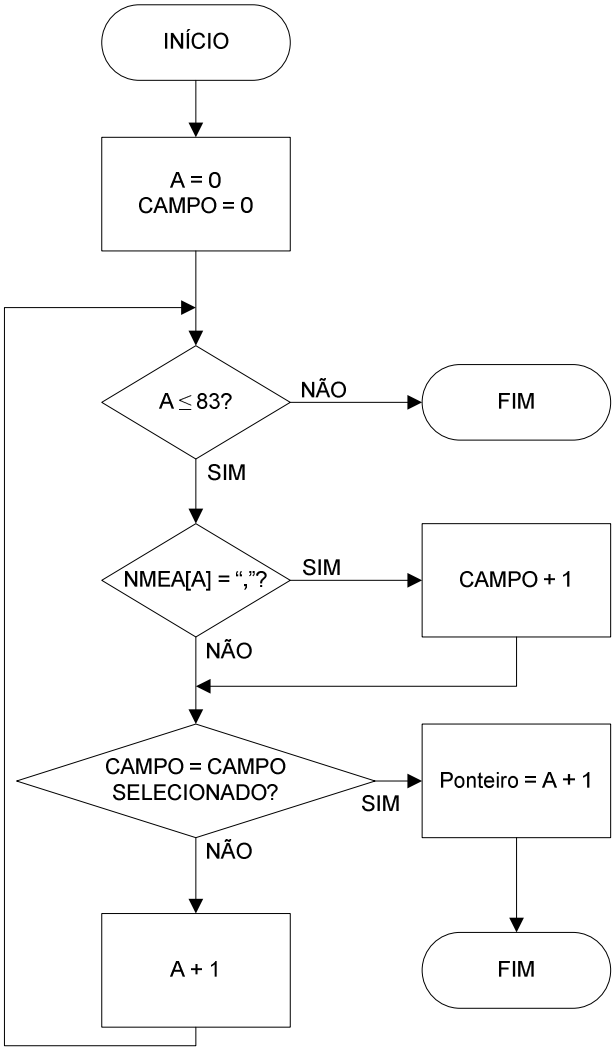
A variável “campo” conta o número do campo atual.

É realizada uma contagem de “a” = 0 até “a” = 83, para analisar todos os caracteres, um a um (linha 4), utilizando-se um laço “for”.

Caso a rotina encontre um caractere “,” (linha5), é somado um ao valor da variável “campo” (linha 6). Desta forma, cada passagem por uma vírgula causa a mudança do campo, como previsto no padrão NMEA 0183.

Se o campo coincidir com o campo selecionado, é atribuído ao ponteiro o valor numérico do caractere atual mais um, ou seja, o ponteiro assume o valor da posição do próximo caractere após a vírgula (linha 14). Ainda na continuação desta condição, o valor de a passa para 83, causando a finalização do laço “for” (linha15).

Se a rotina não encontrar um caractere “,” e nem o campo coincidir com o valor do campo selecionado, o laço “for” incrementa o valor de “a” e recomeça a análise para outro caractere da sentença.



Fonte: própria

Figura 19: fluxograma da atualização do ponteiro

Cada campo tem uma largura de caracteres definida. Sabendo-se onde começa o referido campo dentro da sentença NMEA 0183, podemos extrair os dados da mesma.

Estes campos obedecem a um formato específico para os dados. Cabe lembrar que os dados são apresentados como caracteres ASCII, sendo necessário converter para outro formato adequado ao tipo de dado, como por exemplo, inteiro, ponto flutuante, texto, etc.

O processo de extração dos dados disponibiliza os mesmos em variáveis globais, que podem ser usadas em qualquer parte do programa.

O fluxograma da extração dos dados pode ser observado na figura 20, onde o código fonte ficará assim:

```
1     void extrai_dados(int campo_horario,int campo_latitude,int
campo_longitude, int campo_data)
2     {
3           atualiza_ponteiro(campo_horario);
4           //o ponteiro aponta para o campo das horas
5           if (NMEA[ponteiro]!=',') {
6           //se o caractere após a vírgula não for outra
7           //vírgula, o campo possui dados a serem extraídos
8           //caso o caractere seja vírgula, não ocorre o processamento
9           //desta parte do código
10          //aqui ocorre a extração dos dados com a devida conversão
11          //de tipo para a variável correta.
12          horas=10*(NMEA[ponteiro]-48)+(NMEA[ponteiro+1]-48);
13          minutos=10*(NMEA[ponteiro+2]-48)+(NMEA[ponteiro+3]-48);
14          segundos=10*(NMEA[ponteiro+4]-48)+(NMEA[ponteiro+5]-48);
15          }
16
17          atualiza_ponteiro(campo_latitude);
18          //o ponteiro aponta para o campo da latitude
19          if (NMEA[ponteiro]!=',') {
20          //verifica se o campo está vazio, conforme
21          //modelo anterior
22          lat_graus=10*(NMEA[ponteiro]-48)+(NMEA[ponteiro+1]-48);
23          lat_minutos=100000*(NMEA[ponteiro+2]-
48)+10000*(NMEA[ponteiro+3]-48)+1000*(NMEA[ponteiro+5]-
48)+100*(NMEA[ponteiro+6]-48)+10*(NMEA[ponteiro+7]-48)+(NMEA[ponteiro+8]-
48);
24          lat_minutos=lat_minutos/10000;
25          orient_lat=(NMEA[ponteiro+10]);
26          }
27
28          atualiza_ponteiro(campo_longitude);
29          //o ponteiro aponta para o campo da longitude
30          if (NMEA[ponteiro]!=',') {
31          //verifica se o campo está vazio, conforme
32          //modelo anterior
33          lon_graus=100*(NMEA[ponteiro]-48)+10*(NMEA[ponteiro+1]-
48)+(NMEA[ponteiro+2]-48);
34          lon_minutos=100000*(NMEA[ponteiro+3]-
48)+10000*(NMEA[ponteiro+4]-48)+1000*(NMEA[ponteiro+6]-
```

```
48)+100*(NMEA[ponteiro+7]-48)+10*(NMEA[ponteiro+8]-48)+(NMEA[ponteiro+9]-
48);
35          lon_minutos=lon_minutos/10000;
36          orient_lon=(NMEA[ponteiro+11]);
37          }
38
39          if (campo_data!=99) {
40          //se o campo data é 99, ignora esta parte
41          //(a sentença GGA não possui informação de data)
42          atualiza_ponteiro(campo_data);
43          //o ponteiro aponta para o campo da data
44                if (NMEA[ponteiro]!=',') {
45                //verifica se o campo está vazio, conforme
46                //modelo anterior
47                dia=10*(NMEA[ponteiro]-48)+(NMEA[ponteiro+1]-48);
48                mes=10*(NMEA[ponteiro+2]-48)+(NMEA[ponteiro+3]-48);
49                ano=10*(NMEA[ponteiro+4]-48)+(NMEA[ponteiro+5]-48);
50                }
51          }
52    }
```

O código para executar a atualização do ponteiro funciona da seguinte forma:

Primeiramente, o ponteiro para o campo horário é atualizado (linha 3), conforme visto anteriormente.

Se o caractere apontado pelo ponteiro é uma vírgula (linha 5), o campo não contém dados a serem processados e o mesmo é ignorado, caso contrário, as horas, minutos e segundos são extraídos da sentença NMEA 0183 e armazenados em três variáveis globais do tipo inteiro (linhas 12, 13 e 14), uma para cada dado.

O ponteiro é atualizado para o campo da latitude (linha 17).

Se o caractere apontado pelo ponteiro é uma vírgula (linha 19), o campo não possui dados e é ignorado. Caso contrário, o correspondente da latitude em graus é armazenado em uma variável do global do tipo inteiro (linha 22), os minutos são armazenados em uma variável global do tipo ponto flutuante (linhas 23 e 24) e a orientação da latitude é armazenada em uma variável global do tipo caractere (linha 25).

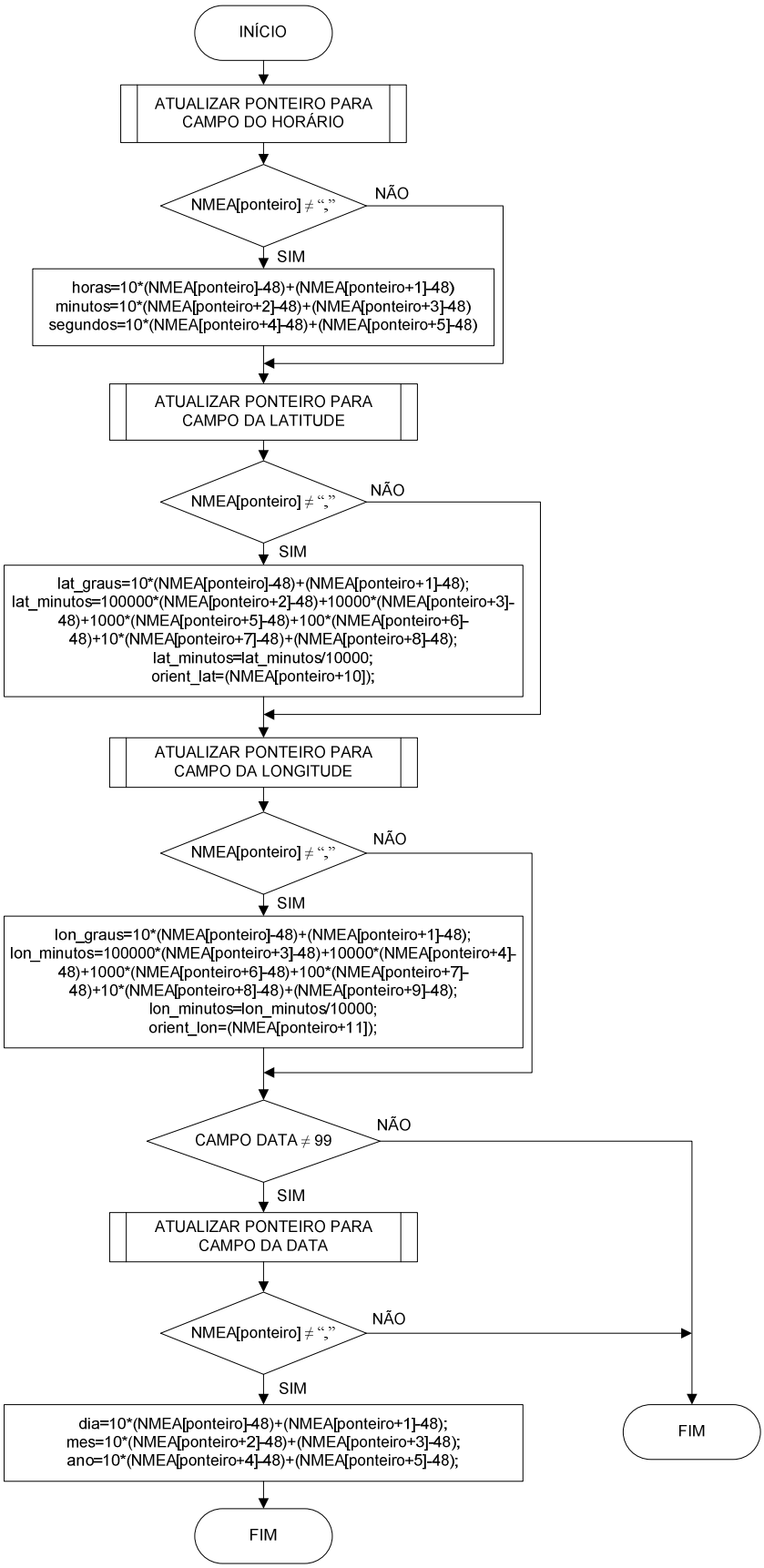
O ponteiro é atualizado para o campo da longitude (linha 28).

Se o caractere apontado pelo ponteiro é uma vírgula (linha 30), o campo não possui dados e é ignorado. Caso contrário, o correspondente da longitude em graus é armazenado em uma variável do global do tipo inteiro (linha 33), os minutos são armazenados em uma variável global do tipo ponto flutuante (linhas 34 e 35) e a orientação da latitude é armazenada em uma variável global do tipo caractere (linha 36).

Para a extração dos dados referentes à data, é verificado se existe a informação de data no campo. Para isso, foi utilizado um código fora dos padrões. Foi escolhido o número 99. Caso o número do campo seja 99, o sistema ignora o campo da data (linha 39) não atualizando o mesmo. Caso contrário é feito o processamento do campo, conforme segue.

O ponteiro é atualizado para o campo da data (linha 42).

Se o caractere apontado pelo ponteiro é uma vírgula, o campo não possui dados e é ignorado. Caso contrário, dia, mês e ano são extraídos da sentença NMEA 0183 e armazenados em três variáveis globais do tipo inteiro (linhas 47, 48 e 49), uma para cada dado



Fonte: própria

Figura 20: fluxograma da extração dos dados

Estando todos os dados disponíveis em variáveis globais, os mesmos podem ser utilizados em qualquer parte do programa. Devido ao fato de muitas aplicações utilizarem cálculo de distância entre duas coordenadas geográficas, foi elaborada uma rotina para o referido cálculo levando em consideração a precisão de uma aplicação de geoposicionamento básica.

9.5. CÁLCULO DA DISTÂNCIA ENTRE DUAS COORDENADAS

Como o receptor de GPS não calcula a distância entre duas coordenadas, foi adicionada à biblioteca tal função, a fim de prover a possibilidade de tal cálculo.

O cálculo entre duas posições geográficas é usado com freqüência, motivo que levou a construção desta parte do código.

Assim, o programador pode obter a distância entre dois pontos como uma função, expandindo as possibilidades do receptor de GPS e não se preocupando em como fazer tal cálculo, desviando sua concentração para a aplicação em si.

Existem muitas formas de calcular a distância entre dois pontos na superfície terrestre, definido por sua latitude e longitude. Os métodos variam em complexidade e precisão.

Conforme [GEOSCIENCE AUSTRALIA], quanto mais simples o método, menos preciso é o cálculo. Os métodos mais conhecidos são:

- Aproximação direta
- Modelo esferoidal da terra
- Distância dos grandes círculos (com base na trigonometria esférica)
- Fórmula de Vincenty (método direto e método inverso)

A aproximação direta consiste em calcular a distância entre a menor das coordenadas e usar como referência. Não é um método preciso, pois não leva em conta que a terra é redonda, também não considera o achatamento dos pólos, muito menos considera as diferenças na latitude e longitude nos diversos pontos do planeta.

Considerando que a menor unidade de medida do nosso GPS é igual a um milésimo de minuto e essa medida equivale a aproximadamente 18 cm em nossa região, podemos usar tranquilamente essa referência para distâncias não superiores

a alguns km, com erro de apenas alguns metros. O erro é desproporcional em todas as direções sendo difícil estimar o mesmo. Pode ser usado apenas como referência local e em sistemas muito simples, baseados em microcontroladores de baixo desempenho.

O modelo esferoidal da terra é mais preciso. A sua precisão é de cerca de 200 metros acima de 50 km, mas pode ficar muito pior em distâncias maiores. Como o cálculo é simples, recomenda-se o uso em sistemas de médio porte e em aplicações locais, como por exemplo, em uma região metropolitana.

Neste modelo, pressupõe-se que a terra é totalmente redonda, com um raio médio de 6.367,963 km. Podem-se usar outros valores que se aproximem mais da realidade de cada região. Os cálculos baseiam-se na trigonometria esférica.

A seqüência de cálculo, conforme [GEOSCIENCE AUSTRALIA] fica assim:

$$Termo1 = 111.08956 \cdot (DL + 0.000001)$$

$$Termo2 = \cos\left[L1 + \left(\frac{DL}{2}\right)\right]$$

$$Termo3 = \frac{(DG + 0{,}000001)}{(DL + 0{,}000001)}$$

$$D = \frac{Termo1}{\cos[\tan^{-1}(Termo2 \cdot Termo3)]}$$

Onde:

DL = latitude do segundo ponto menos a latitude do primeiro ponto, em graus.

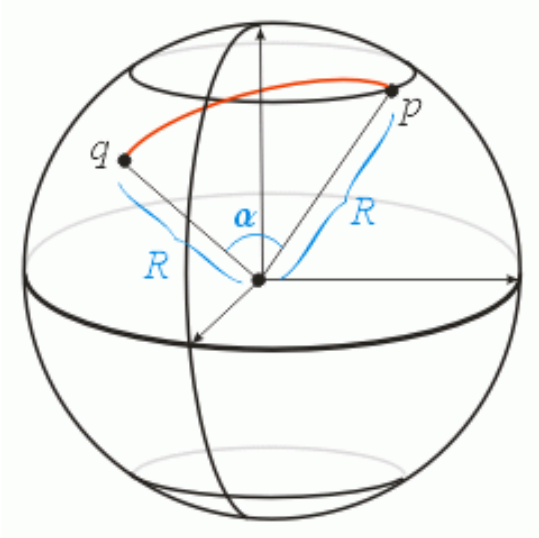
DG = longitude do segundo ponto menos a longitude do primeiro ponto, em graus.

L1 = latitude do primeiro ponto, em graus.

D = distância entre os dois pontos, em km.

Usando-se o modelo esferoidal da terra, o grande problema é que devido o fato de considerar a terra redonda, pode haver muito erro se utilizarmos o cálculo de uma região em outra, principalmente quando mudamos a posição em latitude.

O método da distância dos grandes círculos compreende a distância mais curta entre dois pontos de uma esfera. Isso coincide com a circunferência de um círculo que passa por ambos os pontos, e está fixo no centro da esfera, como se vê na figura 21.



Fonte: James Madison University

Figura 21: Distância dos grandes círculos

Para calcular a distância dos grandes círculos, devemos aplicar a seguinte fórmula, conforme [VENESS]:

$$D = 1{,}852 \cdot 60 \cdot \sin^{-1}[\sin L1 \cdot \sin L2 + \cos L1 \cdot \cos L2 \cdot \cos DG]$$

Onde,

D = distância computada

L1 = latitude do ponto 1

L2 = latitude do ponto 2

DG = longitude do segundo ponto menos a longitude do primeiro ponto

Assumiu-se para efetivação dos cálculos que:

1 minuto de arco = 1 milha náutica

1 milha náutica = 1852 km

Esse método é mais preciso em grandes distâncias e menos preciso para curtas distâncias. Quanto aos cálculos, também é indicado para sistemas de médio porte, assim como o modelo esferoidal da terra, porém apresenta um erro comparativamente menor para distâncias de milhares de km.

As fórmulas de Vincenty são o método mais preciso de cálculo dentre os mais comuns, mas em virtude da sua complexidade matemática, só se recomenda o uso

em sistemas embarcados caso seja fundamental a precisão máxima. O uso de recursos pode inviabilizar um projeto mais complexo.

Normalmente as fórmulas de Vincenty são utilizadas em ambiente computacional, onde existe muito poder de processamento.

Os cálculos usando as fórmulas de Vincenty podem ser extremamente precisos desde a escala milimétrica até dezenas de milhares de km.

Por não se enquadrar neste trabalho, as fórmulas de Vincenty não serão abordadas em profundidade.

Como não será processada uma grande quantidade de dados, foi escolhido o cálculo através do método das distâncias dos grandes círculos. O mesmo poderia garantir uma excelente precisão com um consumo de recursos ideal para o microcontrolador escolhido.

A rotina para o cálculo, usando o método das distâncias dos grandes círculos ficaria assim:

```
1     float distancia_gcd()
2     {
3           float d,lat1,lat2,long1,long2;
4           lat1=(ref_lat_graus+(ref_lat_minutos/60))*pi/180;
5           lat2=(lat_graus+(lat_minutos/60))*pi/180;
6           long1=(ref_lon_graus+(ref_lon_minutos/60))*pi/180;
7           long2=(lon_graus+(lon_minutos/60))*pi/180;
8           d   =   acos(sin(lat1)*sin(lat2)+cos(lat1)*cos(lat2)*cos(long2-
long1))*111120*180/pi;
9           return (d);
10    }
```

O código para executar o cálculo da distância entre duas coordenadas geográficas funciona da seguinte forma:

São inicializadas cinco variáveis do tipo ponto flutuante: “d”, “lat1”, “lat2”, “long1” e “long2” (linha 3). A variável “d” irá armazenar a distância entre as duas coordenadas geográficas. As demais serão usadas para auxiliar a conversão de graus, minutos e frações de minutos para radianos. Isso é feito somando a componente em graus com a componente em minutos dividida por 60. Após, o resultado dessa primeira conta é multiplicado por π e dividido por 180 graus.

A latitude da coordenada 1 é transformada para radianos (linha 4).

A latitude da coordenada 2 é transformada para radianos (linha 5).

A longitude da coordenada 1 é transformada para radianos (linha 6).

A longitude da coordenada 2 é transformada para radianos (linha 7).

Realiza-se o cálculo segundo o método das distâncias dos grandes círculos, com os valores das coordenadas em radianos, armazenando a distância em quilômetros na variável “d” (linha 8).

A função retorna o valor da variável “d” 9.

A função não recebe nenhum parâmetro. Todas as variáveis de entrada são globais. Foram criadas variáveis locais para auxiliar nos cálculos. Dessa forma, ao invés de ter uma equação enorme, teremos cinco equações menores.

A função retorna o valor do cálculo da distância entre o ponto atual e o ponto marcado previamente como referência.

Para fazer com que as coordenadas atuais sejam usadas como referência para o cálculo, pode-se usar a função:

```
1     void referencia()
2     {
3           ref_lat_graus=lat_graus;
4           ref_lon_graus=lon_graus;
5           ref_lat_minutos=lat_minutos;
6           ref_lon_minutos=lon_minutos;
7     }
```

A função utilizada para fixar uma coordenada geográfica como referência copia os valores de latitude e longitude para outras variáveis, que são utilizadas no cálculo da distância dos grandes círculos.

No anexo A pode-se observar a biblioteca completa.

10. RESULTADOS

Para validar a biblioteca e os cálculos, foi feito um programa em C, no compilador Borland TURBO C++ 2008. Foram utilizadas sentenças NMEA 0183 adquiridas do receptor GPS descrito no capítulo 6.

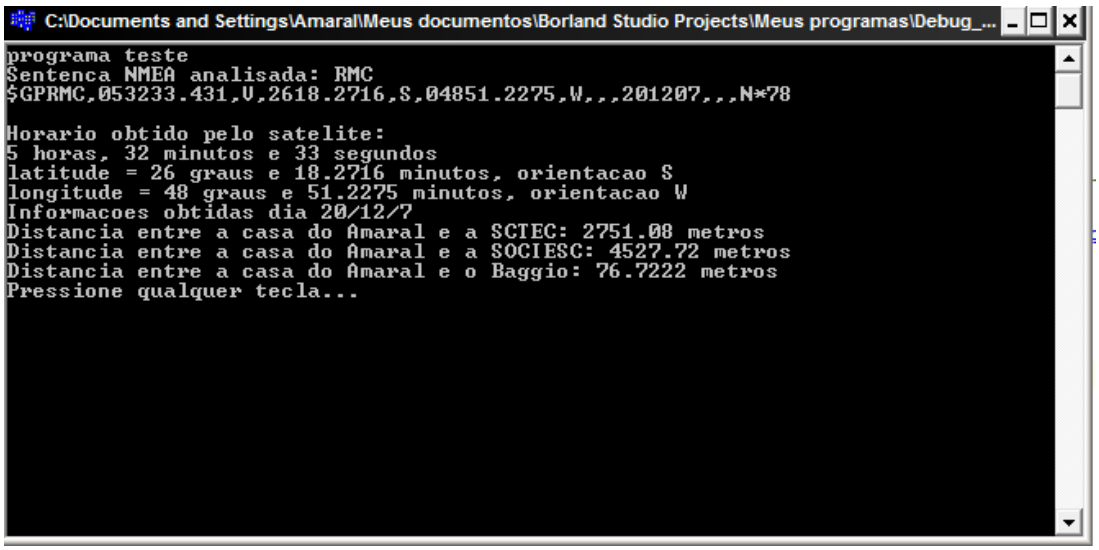
As sentenças foram selecionadas de tal forma que todas as funcionalidades da biblioteca fossem testadas.

O controle foi elaborado assim: para as sentenças GGA, uma sentença com um caractere trocado propositalmente, uma sentença sem o campo de checksum, uma sentença com um campo faltante, uma sentença totalmente vazia, uma sentença completa e funcional. Para as sentenças RMC foi adotado o mesmo procedimento. No total, foram testadas 10 sentenças diferentes no compilador.

Como as sentenças foram informadas no código do programa, o resultado da análise era previsível.

No mesmo programa, para validar a precisão dos cálculos, foi inserido como referência duas posições geográficas conhecidas e o resultado do cálculo foi comparado à medição realizada no software *Google Earth*, obtido em http://earth.google.com/intl/pt/.

O resultado do teste pode ser visto na figura 22.

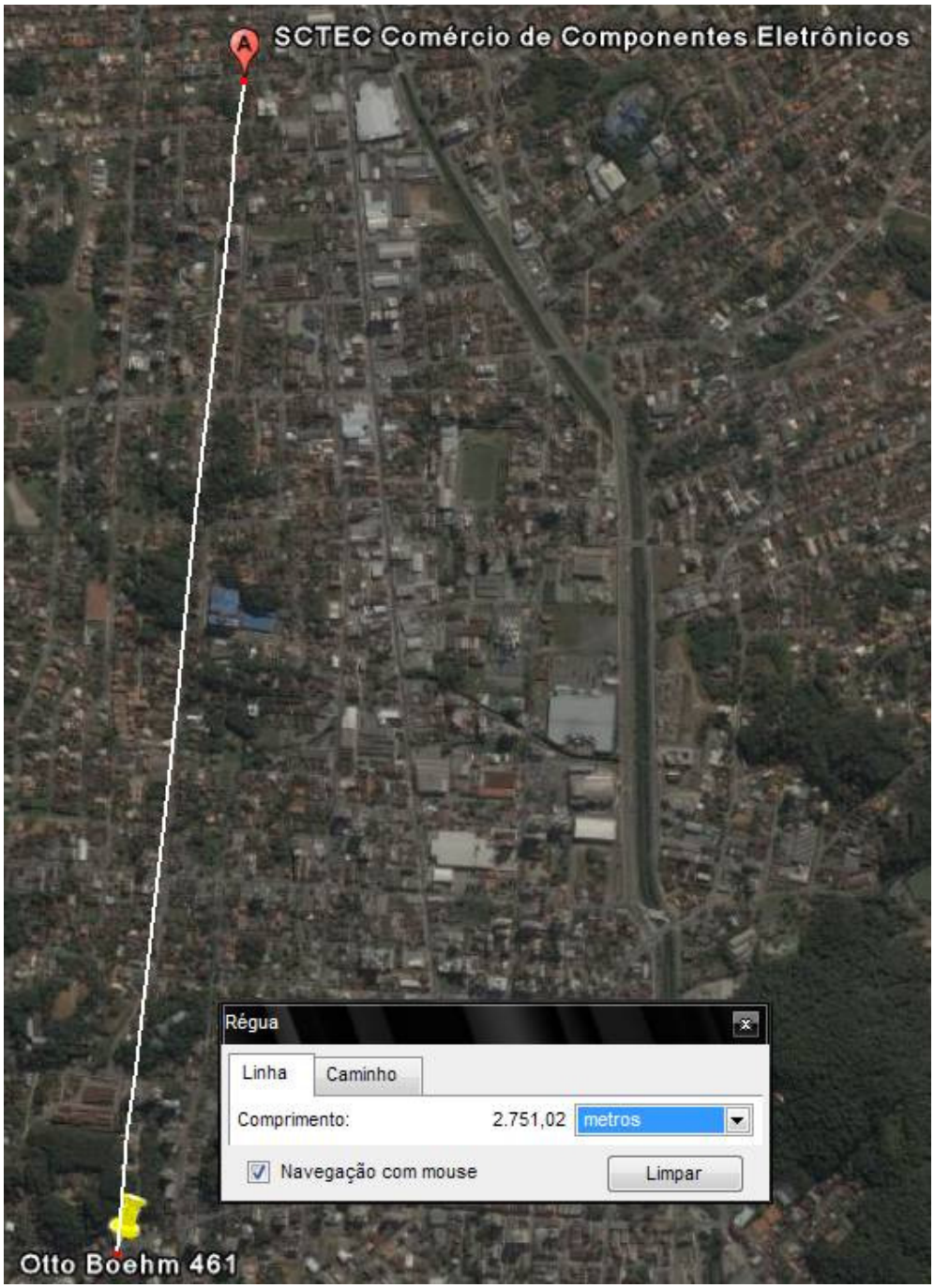


Fonte: própria

Figura 22: Resultado dos testes

Em três medições, o sistema se mostrou altamente confiável, com erro menor que a resolução do software *Google Earth*, justificando a diferença de casas decimais das medições.

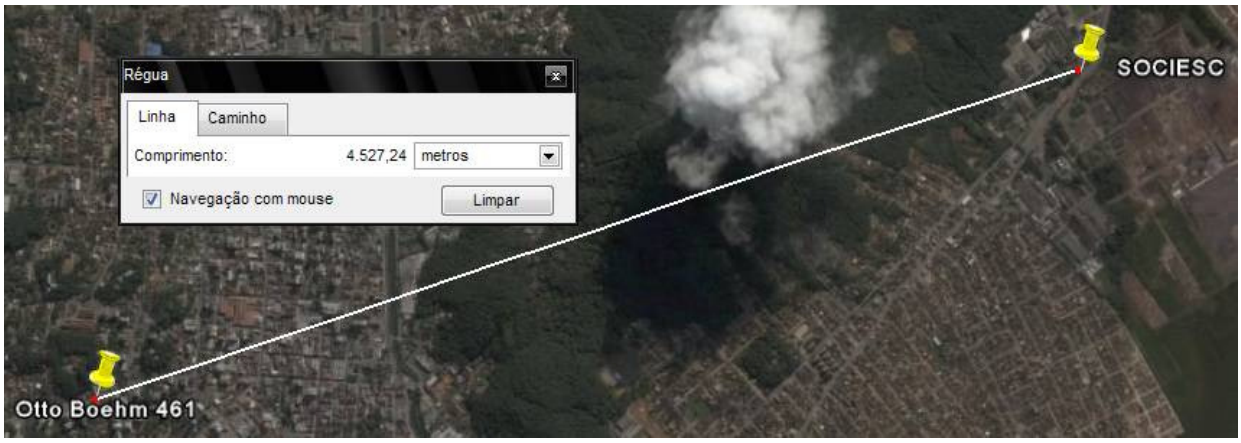
No primeiro cálculo, usou-se como referência a casa do autor e o escritório da empresa SCTEC. A figura 23 mostra o resultado da medição através do Google Earth, que pode ser comparado à figura 22.



Fonte: própria

Figura 23: resultado dos testes

A segunda medição tomou como referência a casa do autor e as instalações da SOCIESC, em uma distância maior. Novamente o erro foi desprezível. Pode-se observar o resultado da análise com o *Google Earth* na figura 24.



Fonte: própria

Figura 24: resultado dos testes

Em um último teste, foi tomado como referência um ponto muito próximo, distante apenas alguns metros. Analisando o resultado através do *Google Earth*, na figura 25, percebe-se que não houve erro mensurável.

O teste de análise das sentenças e extração dos dados foi concluído com sucesso.

A validação do funcionamento da biblioteca foi obtida com a realização dos testes a seguir:

Sentenças com caracteres trocados: foram detectadas e descartadas;

Sentenças com campos faltantes: foram detectadas e descartadas;

Sentenças sem o campo de checksum: foram detectadas e descartadas;

Sentenças vazias: foram interpretadas, sem alteração dos dados;

Sentenças válidas: tiveram seus dados extraídos.

Após a validação da biblioteca, a mesma foi testada nas plataformas com microcontroladores.

Atualmente a biblioteca é usada comercialmente pela empresa SCTEC em projetos com eletrônica embarcada. Um dos projetos que fazem uso da biblioteca

(até a última atualização deste trabalho) é um equipamento médico, que usa as coordenadas obtidas através do receptor GPS para criar uma cerca geográfica. Este equipamento funciona apenas na área onde foi habilitado, a pedido do cliente.

Fonte: própria

Figura 25: resultado dos testes

11. CONCLUSÃO

Com a biblioteca para GPS, torna-se muito mais fácil desenvolver projetos utilizando tal tecnologia, pois a preocupação do projetista passa a ser a aplicação em si, e não a programação das rotinas de GPS.

Uma vantagem de ter uma biblioteca única para diversas plataformas é o fato de poder migrar de uma para outra, em função da necessidade ou custo.

Se o projeto começar com uma aplicação pequena e sofrer uma atualização que demande na utilização de outra plataforma, o código não será alterado nas rotinas do GPS, diminuindo o custo das horas de programação e acelerando o processo.

Outra grande vantagem é a facilidade de uso, visto que não é necessário estudar e aprender vários comandos diferentes para as plataformas, pois a biblioteca é uma só.

Uma desvantagem aparece na plataforma mais simples. Os microcontroladores de 8 bits normalmente têm pouca memória. Para conseguir compatibilidade total em todas as plataformas, sem qualquer alteração no código, foi necessário usar os tipos de variáveis que funcionassem em todas as plataformas. Isso aumentou o consumo de memória de maneira desprezível para as plataformas maiores, mas foi significativo nos microcontroladores de 8 bits.

Uma dificuldade encontrada na elaboração das rotinas, em virtude das diferentes plataformas utilizadas foi a declaração das variáveis do sistema.

Uma rotina em especial, a da validação da sentença NMEA 0183, seria idealmente declarada como tipo booleano, por se tratar de uma rotina que retorna apenas os valores “verdadeiro” ou “falso”.

No compilador EWARM, para a plataforma ARM, não existe o tipo booleano. O menor tamanho de variável é do tipo *char*, que ocupa 8 bits, ou uma posição de memória inteira.

Para que a rotina fosse compatível com todas as plataformas, a mesma foi declarada como *char* (caractere ASCII).

Um ponto interessante a se observar no projeto é que apesar de ser totalmente possível a implementação de rotinas com GPS em microcontroladores de

pequeno porte, deve-se observar a total quantidade de memória RAM. No nosso caso, o microcontrolador PIC 16F877 possui 386 bytes de memória RAM, dos quais, mais de 2/3 foram usados apenas para as rotinas envolvendo GPS.

Para algumas aplicações, o uso de memória pode ser crítico, forçando o projetista a utilizar uma memória externa ou outro tipo de microcontrolador.

Quanto à velocidade de execução das rotinas, não houve nenhum impedimento nem se verificou nenhuma atenção especial. Os cálculos matemáticos foram escolhidos criteriosamente para atender ao menor hardware escolhido e a uma distância razoável para a maioria das aplicações em localização e controle de patrimônio ou pessoas.

Os cálculos apresentam erros pequenos até mesmo em milhares de quilômetros, conforme [GEOSCIENCE AUSTRALIA].

Experimentalmente, nenhum erro de cálculo foi observado. Os testes envolveram lugares conhecidos e pontos de até 10 quilômetros. Para realizar os testes, foram inseridas coordenadas no código, a fim de simular uma medição. Os resultados foram comparados com medições reais no software *Google Earth*, que retornou as mesmas medidas.

Sugere-se a continuidade deste trabalho aperfeiçoando a biblioteca ou criando projetos completos utilizando GPS, visto que as bibliotecas estão disponíveis neste trabalho.

O código foi planejado para ser flexível. Em se havendo a necessidade de outras informações oriundas das sentenças NMEA 0183, sugere-se seguir os roteiros e fluxogramas propostos no presente trabalho, a fim de aperfeiçoar a biblioteca.

12. GLOSSÁRIO

ARM – *advanced risc machines* (máquinas avançadas RISC).

ASCII – *American Standard Code for Information Interchange* (Código Padrão Americano para o Intercâmbio de Informação).

CR – *Carriage Return* (retorno de carro).

DoD – *Department of Defense* (Departamento de Defesa dos Estados Unidos).

DOP – Dilution of Precision (diluição da precisão).

EIA – *Electronic Industries* Alliance (Aliança das Indústrias Eletrônicas dos Estados Unidos).

GDOP – Geometric DOP (DOP geométrico).

GPS – *global positioning system* (sistema de posicionamento global).

HDOP – Horizontal DOP (DOP horizontal).

I2C – Inter-Integrate Circuit (Circuito Inter-Integrado).

LF – *Line Feed* (alimentação de linha).

NAVSTAR – *Navigation Satellite with Time and Ranging* (Sistema de navegação com tempo e alcance).

NMEA – National Marine Electronics Association (Associação Nacional de Eletrônica Naval dos Estados Unidos).

OEM – Original Equipment Manufacturer (fabricante de equipamento original).

PDOP – Position (3-D) DOP (DOP posicional ou tridimensional).

RAM – *Random Access Memory* (memória de acesso aleatório).

RISC – *reduced instruction set computer* (computador com set de instruções resumido).

SPI – *Serial Peripheral Interface* (Interface Serial para Periféricos).

TDOP – Time DOP (DOP temporal).

USART – *Universal Synchronous Asynchronous Receiver Transmitter* (Transmissor/Receptor Universal Síncrono e Assíncrono).

USB – *Universal Serial Bus* (Barramento Serial Universal).

UTC – *Universal Time, Coordinated* (Tempo Universal, coordenado).

VDOP – Vertical DOP (DOP vertical).

13. REFERÊNCIAS BIBLIOGRÁFICAS

CÉGEP DE DRUMMONDVILLE. **The extended ASCII Chart**. Disponível em <http://www.cdrummond.qc.ca/cegep/informat/Professeurs/Alain/files/ascii.htm>. acesso em 14 de maio de 2008.

GARMIN CORPORATION. **GPS Guide for Beginners**. Estados Unidos. 2000.

GARMIN CORPORATION. **What is GPS?** Disponível em: <http://www8.garmin.com/aboutGPS/>. Acesso em 4 de maio de 2008.

GEOSCIENCE AUSTRALIA – AUSTRALIAN GOVERNMENT. **Calculating Distance Between Two Points**. Updated 19 april 2007. Disponível em <http://www.ga.gov.au/geodesy/datums/distance.jsp>. Acesso em 4 de maio de 2008.

GORGULHO, Miguel. **G.P.S. - O "Sistema de Posicionamento Global"** Disponível em <http://www.gpsglobal.com.br/Artigos/Apostila.html#23>. Acesso em 4 de maio de 2008.

GPSINFORMATION.NET. Diversos Artigos. Disponível em <http://gpsinformation.net/>. Acesso em 4 de maio de 2008.

GURGACZ, G.; NASCIMENTO, Z. **Metodologia do Trabalho Científico com Enfoque nas Ciências Exatas**. Joinville. SOCIESC, 2007. 132 p.

INFO EXAME. São Paulo. Editora Abril. v.266. ano 22. Abr. 2008. Edição Especial GPS.application

KOWOMA. **GPS – Explained**. Sources of Errors in GPS. Disponível em <http://www.kowoma.de/en/gps/errors.htm>. Acesso em 4 de maio de 2008.

MIYOSHI, E.; SANCHES, C.; **Projetos de Sistemas Rádio**. São Paulo. Editora Érica. 2002. 534 p. ISBN 85-7194-868-2

PEREIRA, Fábio. **Microcontroladores PIC**: Programação em C. São Paulo. Editora Érica. 2003. 358 p. ISBN 85-7194-935-2

PEREIRA, Fábio. **Tecnologia ARM**: Microcontroladores de 32 Bits. São Paulo. Editora Érica. 2007. 448 p. ISBN 978-85-365-0170-3

PESTANA, Antônio. **Sistema de Posicionamento Global**: NAVSTAR/GPS. Portugal. INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO. 27 p. 2002.

ROCHA, José Antônio M. R. **GPS – Uma abordagem prática**: Para Aplicações Náuticas Terrestres e de Geoprocessamento. Recife. Edições Bagaço, 2003. 235 p. ISBN 85-7409-119-7

SCHILDT, Herbert. **C Completo e Total**. São Paulo. Makron Books. 1996. 856 p. ISBN 85-346-0595-5

TRIMBLE. **Trimble GPS Tutorial**. GPSTutorial.exe. Sunnyvale. Estados Unidos. 4 de maio de 2008. 1 arquivo. (6.117 kBytes). Arquivo eletrônico auto executável.

UNIVERSIDADE FEDERAL DO PARANÁ. **Programas/Arquivos – Código ASCII**. Disponível em <http://www2.ufpa.br/dicas/progra/arq-asc.htm>. Acesso em 14 de maio de 2008.

VENESS, Chris. **Geodesic distance between two Latitude/Longitude points using Vincenty ellipsoid formula in JavaScript**. Disponível em: <http://www.movable-type.co.uk/scripts/latlong-vincenty.html>. Acesso em 4 de maio de 2008.

VINCENTY, T. Direct and Inverse Solutions of Geodesics and Ellipsoid With Application of Nested Equations. **SURVEY REVIEW**. Tolworth, Inglaterra. v.23, n.176, p.88-93, abr. 1975.

WAKERLY, J.F. **Error detecting Codes, Self-Checking Circuits and Applications**. North-Holland. Elsevier, Inc. 1978. 231p.

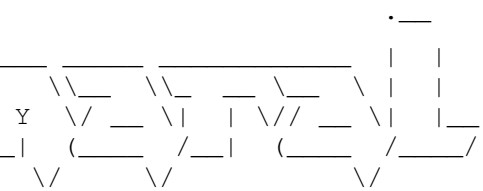
14. ANEXOS

ANEXO A: BIBLIOTECA PARA EXTRAÇÃO DE DADOS GPS (PARSER)

ANEXO B: PROGRAMA TESTE PARA VALIDAÇÃO DA BIBLIOTECA

ANEXO A: BIBLIOTECA PARA EXTRAÇÃO DE DADOS GPS (PARSER)

```
/*
Parser para GPS padrão NMEA sentenças GGA e RMC
 Dezembro de 2007 Versão 1.0 Revisão 0
 Desenvolvido para SCTEC - Automação e Projetos Especiais por
 ROBERTO DO AMARAL SALES
```



```
 Resumo:
        Este código extrai dados relativos à hora, minuto e segundo (UTC),
        latitude em graus, minutos e fração de minuto, orientação da latitude,
        longitude em graus, minutos e fração de minuto, orientação da longitude,
        dia, mês e ano, de uma sentença NMEA padrão GGA ou RMC, conforme estes
        dados estejam disponíveis.

        Este código fixa ainda uma referência geográfica para posteriormente
        calcular a distância entre a mesma e outra coordenada qualquer.
        O código foi criado para funcionar com o sistema padrão NMEA,
        aceitando latitude em graus e frações de minutos e longitude em
        graus e frações de minutos.

        Os dados ausentes não são sobrescritos em um processo de atualização, ou
        seja, o dado contido nas variáveis é o último dado válido, obtido em uma
        sentença NMEA.

 As variáveis globais usadas são as seguintes:

      char NMEA[84]          - string NMEA obtida pela porta serial,
      int ponteiro           - posição do caractere na sentença NMEA
      int horas
      int minutos
      int segundos
      int lat_graus
      int lon_graus
      int dia
      int mes
      int ano
      float lat_minutos
      float lon_minutos
```

```
       char orient_lat
       char orient_lon
       int ref_lat_graus
       int ref_lon_graus
       float ref_lat_minutos
       float ref_lon_minutos
       float d                  - distância entre as duas coordenadas
       const float pi           - o número pi

 Uso: Para atualizar todas as variáveis, use a função atualiza(void)

/*------------------------ BIBLIOTECAS USADAS -----------------------------*/

#include <math.h>

/*---------------------------- CONSTANTES ---------------------------------*/

       const float pi=3.1415926535897932385;

/****************** FIXAR REFERÊNCIA NA COORDENADA ATUAL ****************/

void referencia()
{
       ref_lat_graus=lat_graus;
       ref_lon_graus=lon_graus;
       ref_lat_minutos=lat_minutos;
       ref_lon_minutos=lon_minutos;
}

/************************ CÁLCULO DA DISTÂNCIA ***************************/

float distancia_gcd()
{
       float d;
       float lat1,lat2,long1,long2;
       lat1=(ref_lat_graus+(ref_lat_minutos/60))*pi/180;
       lat2=(lat_graus+(lat_minutos/60))*pi/180;
       long1=(ref_lon_graus+(ref_lon_minutos/60))*pi/180;
       long2=(lon_graus+(lon_minutos/60))*pi/180;
       d = acos(sin(lat1)*sin(lat2)+cos(lat1)*cos(lat2)*cos(long2-
long1))*111120*180/pi;
       return (d);
}

/************************ VALIDAÇÃO DA STRING ****************************/
```

```
boolean string_valida()
{
	int checksum,string,a=0;
	char checksum_string[3];
	do{
	a++;
	}while (NMEA[a]!='*'&&a<=83&&NMEA[a]!=NULL);
	if (NMEA[a]=='*') {
		memset(checksum_string,0,3);
		checksum_string[0]=NMEA[a+1];
		checksum_string[1]=NMEA[a+2];
		checksum=strtol(checksum_string,0,16);
		a=1; //começa do segundo caractere
		string=(NMEA[a]);
		while (NMEA[a+1]!='*'){
		a++;
		string=(string^NMEA[a]);
		}
		if (string==checksum) {
			return true;
		}
		else
		return false;
	}
	else
	return false;
}

/********************** DETERMINAÇÃO DOS PONTEIROS ***********************/

void atualiza_ponteiro(int campo_selecionado)
{
	int a,campo=0;			//começa em campo = 0
	for (a=0; a <= 83; a++) {	//conta de 0 a 83
		if (NMEA[a]==',') {	//se um dos caracteres é vírgula,
		campo++;		//soma um na variável campo
		}
		if (campo==campo_selecionado) { //se a variável campo coincide com
		ponteiro=a+1;		//a variável escolhida, o próximo caractere
		a=83;			//é o ponteiro para o dado a ser extraído e
		}			//termina a contagem, retornando para a função
	}				//que chamou esta
}

/******************** EXTRAÇÃO DOS DADOS DA SENTENÇA NMEA *****************/
```

```
void extrai_dados(int campo_horario,int campo_latitude,int campo_longitude,int
campo_data)
{
/*-------- EXTRAÇÃO DAS HORAS, MINUTOS E SEGUNDOS DA SENTENÇA NMEA --------*/

       atualiza_ponteiro(campo_horario);

       if (NMEA[ponteiro]!=',') {
              horas=10*(NMEA[ponteiro]-48)+(NMEA[ponteiro+1]-48);
              minutos=10*(NMEA[ponteiro+2]-48)+(NMEA[ponteiro+3]-48);
              segundos=10*(NMEA[ponteiro+4]-48)+(NMEA[ponteiro+5]-48);
       }

/*------------- EXTRAÇÃO DA LATITUDE EM GRAUS, MINUTOS E ORIENTAÇÃO --------*/

       atualiza_ponteiro(campo_latitude);

       if (NMEA[ponteiro]!=',') {
       lat_graus=10*(NMEA[ponteiro]-48)+(NMEA[ponteiro+1]-48);
       lat_minutos=100000*(NMEA[ponteiro+2]-48)+10000*(NMEA[ponteiro+3]-
48)+1000*(NMEA[ponteiro+5]-48)+100*(NMEA[ponteiro+6]-48)+10*(NMEA[ponteiro+7]-
48)+(NMEA[ponteiro+8]-48);
       lat_minutos=lat_minutos/10000;
       orient_lat=(NMEA[ponteiro+10]);
       }

/*----------- EXTRAÇÃO DA LONGITUDE EM GRAUS, MINUTOS E ORIENTAÇÃO ---------*/

       atualiza_ponteiro(campo_longitude);

       if (NMEA[ponteiro]!=',') {
       lon_graus=100*(NMEA[ponteiro]-48)+10*(NMEA[ponteiro+1]-48)+(NMEA[ponteiro+2]-
48);
       lon_minutos=100000*(NMEA[ponteiro+3]-48)+10000*(NMEA[ponteiro+4]-
48)+1000*(NMEA[ponteiro+6]-48)+100*(NMEA[ponteiro+7]-48)+10*(NMEA[ponteiro+8]-
48)+(NMEA[ponteiro+9]-48);
       lon_minutos=lon_minutos/10000;
       orient_lon=(NMEA[ponteiro+11]);
       }

/*--------------------------- EXTRAÇÃO DA DATA ----------------------------*/

       if (campo_data!=99) {

              atualiza_ponteiro(campo_data);
```

```
            if (NMEA[ponteiro]!=',') {
                  dia=10*(NMEA[ponteiro]-48)+(NMEA[ponteiro+1]-48);
                  mes=10*(NMEA[ponteiro+2]-48)+(NMEA[ponteiro+3]-48);
                  ano=10*(NMEA[ponteiro+4]-48)+(NMEA[ponteiro+5]-48);
            }
      }
}

/************************ ATUALIZAÇÃO DOS DADOS ***************************/

void atualiza()
{
      if (string_valida()) {
            if (NMEA[2]=='R'&&NMEA[3]=='M'&&NMEA[4]=='C') {
                  //campo_horario = 1
                  //campo_latitude = 3
                  //campo_longitude = 5
                  //campo_data = 9
                  extrai_dados(1,3,5,9);
            }
            if (NMEA[2]=='G'&&NMEA[3]=='G'&&NMEA[4]=='A') {
                  //campo_horario = 1
                  //campo_latitude = 2
                  //campo_longitude = 4
                  //campo_data = 99 (inválido)
                  extrai_dados(1,2,4,99);
            }
      }
}
```

ANEXO B: PROGRAMA TESTE PARA VALIDAÇÃO DA BIBLIOTECA

```
/*-------------------------------------------------------------------------*//*
_________ _______  _______ _________ _______ 
\__   __/(  ____ \(  ____ \\__   __/(  ____ \
   ) (   | (    \/| (    \/   ) (   | (    \/
   | |   | (__    | (_____    | |   | (__    
   | |   |  __)   (_____  )   | |   |  __)   
   | |   | (            ) |   | |   | (      
   | |   | (____/\/\____) |   | |   | (____/\
   )_(   (_______/\_______)   )_(   (_______/

 _______  _______  _______  _______  _______  _______ 
(  ____ )(  ___  )(  ____ )(  ____ \(  ____ \(  ____ )
| (    )|| (   ) || (    )|| (    \/| (    \/| (    )|
| (____)|| (___) || (____)|| (_____ | (__    | (____)|
|  _____)|  ___  ||     __)(_____  )|  __)   |     __)
| (      | (   ) || (\ (         ) || (      | (\ (   
| )      | )   ( || ) \ \__/\____) || (____/\| ) \ \__
|/       |/     \||/   \__/\_______)(_______/|/   \__/

*//*------------------------ VARIÁVEIS GLOBAIS -------------------------------*/

       int ponteiro, horas, minutos, segundos, lat_graus, lon_graus, dia, mes, ano,
ref_lat_graus, ref_lon_graus;
       float lat_minutos,lon_minutos,ref_lat_minutos,ref_lon_minutos;
       char orient_lat,orient_lon;

/*------------------------- STRINGS DE TESTE ------------------------------*/
//Para testar, remover o comentário da frente de uma das sentenças

//char NMEA[84] = "$GPRMC,003241.000,A,2618.2191,S,04851.2564,W,2.70,”;
//char NMEA[84] = "$GPGGA,003242.000,2618.2166,S,04851.2578”;
//char NMEA[84] = "$GPRMC,003023.436,V,,,,,,,201207,,,N*48”;
//char NMEA[84] = "$GPGGA,003024.436,,,,,0,00,,,M,0.0,M,,0000*52”;
char NMEA[84] = "GPRMC,053233.431,V,2618.2716,S,04851.2275,W,,,201207,,,N*78";
//char NMEA[84] =
"GPRMC,003241.000,A,2618.2191,S,04851.2564,W,2.70,331.25,201207,,,A*6B";
//char NMEA[84] = "GPRMC,003027.438,V,,,,,,,201207,,,N*42";
//char NMEA[84] = "GPGGA,003042.431,,,,,0,00,,,M,0.0,M,,0000*55";
//char NMEA[84] =
"GPGGA,003240.000,2618.2215,S,04851.2546,W,1,03,9.0,113.1,M,0.9,M,,0000*63";
//char NMEA[84] =
"$GPGGA,003246.000,2618.2160,S,04851.2383,W,6,00,50.0,107.5,M,0.9,M,,0000*54”;

/*-------------------------------------------------------------------------*/

#include <vcl.h>
#include <stdio.h>
#include <stdlib.h>
```

```
#include <conio.h>
#include <C:\Documents and Settings\Amaral\Meus documentos\Borland Studio
Projects\GPS\parser.h>
#pragma hdrstop

/*************************** BLOCO PRINCIPAL ****************************/

int main()
{
/*------------------------------ USO DO PARSER -----------------------------*/

      atualiza();  //verifica o buffer de recepção da sentença NMEA.
                           //Se o dado for válido, extrai os dados conforme
                           //a disponibilidade dos mesmos.

/*-------------------------- VERIFICAÇÃO DOS DADOS -------------------------*/

      printf("programa teste\n");
      printf("Sentenca NMEA analisada: %C%C%C \n$%s
\n\n",NMEA[2],NMEA[3],NMEA[4],NMEA);
      printf("Horario obtido pelo satelite: \n");
      printf("%d horas, %d minutos e %d segundos\n",horas,minutos,segundos);
      printf("latitude = %d graus e %G minutos, orientacao
%c\n",lat_graus,lat_minutos,orient_lat);
      printf("longitude = %d graus e %G minutos, orientacao
%c\n",lon_graus,lon_minutos,orient_lon);
      printf("Informacoes obtidas dia %d/%d/%d\n",dia,mes,ano);

/*---------------------------- MEDIDA DE DISTÂNCIA -------------------------*/

/*---------------------------- PONTO DE REFERÊNCIA -------------------------*/

// 26° 18.139'S  LATITUDE NA MINHA CASA
      lat_graus=26;
      lat_minutos=18.139;
// 48° 51.321'O  LONGITUDE NA MINHA CASA
      lon_graus=48;
      lon_minutos=51.321;

/*---------------------------- FIXANDO A REFERÊNCIA ------------------------*/

      referencia();  //aplica o ponto atual como referência para o sistema
                           //de coordenadas. A partir deste ponto será calculada
                           //a distância para os demais.

/*------------------------------ SEGUNDO PONTO -----------------------------*/
```

```
//       26° 16.662'S  LATITUDE NA SCTEC
       lat_graus=26;
       lat_minutos=16.662;
//       48° 51.144'O  LONGITUDE NA SCTEC
       lon_graus=48;
       lon_minutos=51.144;

/*------------------------- VERIFICAÇÃO DOS DADOS -------------------------*/

       printf("Distancia entre a casa do Amaral e a SCTEC: %g
metros\n",distancia_gcd());

/*----------------------------- OUTRO EXEMPLO -----------------------------*/

//      26° 17.369'S  LATITUDE NA SOCIESC
       lat_graus=26;
       lat_minutos=17.369;
//      48° 48.733'O  LONGITUDE NA SOCIESC
       lon_graus=48;
       lon_minutos=48.733;

/*------------------------- VERIFICAÇÃO DOS DADOS -------------------------*/

       printf("Distancia entre a casa do Amaral e a SOCIESC: %g
metros\n",distancia_gcd());

/*--------------------------- MAIS OUTRO EXEMPLO ---------------------------*/

//       26° 18.117'S  LATITUDE NO BAGGIO
       lat_graus=26;
       lat_minutos=18.117;
//       48° 51.282'O LONGITUDE NO BAGGIO
       lon_graus=48;
       lon_minutos=51.282;

/*------------------------- VERIFICAÇÃO DOS DADOS -------------------------*/

       printf("Distancia entre a casa do Amaral e o Baggio: %g
metros\n",distancia_gcd());

       printf("Pressione qualquer tecla...");
       getch();
       return 0;
}
```