Universidade do Minho

Escola de Engenharia

Departamento de Electrónica Industrial

# Aquisição Multisensorial e Controlo de uma Plataforma Móvel

**Luís Fernando Costa Pacheco**

Dissertação submetida à Universidade do Minho para obtenção do grau de Mestre em Electrónica Industrial e Computadores

Novembro, 2007

Dissertação realizada sob a orientação cientifica do
Professor António Fernando Macedo Ribeiro
Professor associado do Departamento de Electrónica
Industrial da Universidade do Minho

“Se é muito difícil encontrar o caminho, faça-o.”

(Autor desconhecido)

# Resumo

Sendo Portugal um pais vocacionado para o turismo, devido às características geográficas que apresenta, faz do golfe mais uma oportunidade para ajudar o seu crescimento. Temos na Europa o terceiro maior mercado do mundo, com cerca de 6 milhões de jogadores. Não só como ponto turístico, é importante salientar, a importância para a economia, porque temos cerca de 2340 postos de trabalho directo, e cerca de 350 milhões de euros por ano em receitas turísticas, comércio de equipamentos, maquinaria e produtos de manutenção, hotéis, restauração, animação, aluguer de automóveis entre muitos outros.

Os treinos num jogo de golfe são essenciais. Existem alguns campos onde se podem treinar todas as situações, desde "*bunker*" (areia), "*putt in green*" (última tacada para o buraco) e "*tee-shot*" (tacada de saída) com o nome de "*driving range*". Estes três pontos são extremamente importantes num jogo de golfe.

Note-se que, um jogador pode usar cerca de 50 bolas em menos de 45 minutos para treinar o "*tee-shot*", o que implica que elas vão ter que ser recolhidas posteriormente.

Este tipo de campo de treino apresenta então alguns problemas, tais como a necessidade de um número elevado de bolas, tempo de paragem da prática aquando da recolha das mesmas para praticar a tacada de saída, entre outros.

Esta dissertação descreve o desenvolvimento de um robô para realizar a recolha de bolas de golfe, num campo de treinos, operado de modo autónomo e/ou remotamente. Salienta-se o uso de diversos sensores para o dotar de capacidades de percepção do ambiente que o rodeiam, assim como um receptor de GPS para realizar o seu posicionamento e auxiliar a navegação no espaço envolvente.

Sendo um protótipo, este trabalho envolveu a criação e manutenção de uma estrutura mecânica, electrónica e software adequados para realizar a tarefa pretendida.

O trabalho realizado foi bem sucedido, porque o protótipo está a ser usado num campo de treinos de praticas de golfe. Foi ainda submetida uma patente ao instituto nacional da propriedade industrial (INPI).

**Palavras-chave** – robô autónomo, golfe, sensores, localização, GPS.

# Abstract

The geographical location of Portugal makes it a great destiny for holidays and tourism, and their golf courses are becoming very popular and huge attractions. The country is dedicating some attention to that new industry bringing about its development.

Europe has the third largest market of golf in the world, with about 6 millions players. It is also important to point out its relevance to the economy not just for the golf itself but also for the number of employees directly or indirectly involved. Around about 2340 direct jobs, and some 350 millions euros per year in tourist revenues, equipment trade, machinery and products for the maintenance, hotels, catering, entertainment, rental cars, among many others.

Training is essential for game success. There are some field areas where one can train all game situations, like "bunker" (sand), "putt in green" (last hit to the hole) and "tee-shot" (start hit) with the name of "driving range". Each of these three points are extremely and equally important in a golf game.

Such training spaces have problems like the huge number of golf balls in stock, the stoppage time for picking up the balls, amongst others. On average a player can hit around about 50 balls in just under 45 minutes to train the "tee-shot".

This thesis describes the work carried out to develop a robot to perform the golf balls picking up in a training field, whether autonomous or remotely operated. Also important to note is the use of multiple sensors to provide the surroundings perception capabilities, and a GPS receiver to perform its positioning and navigation.

The prototype was built from scratch, and included the development and maintenance of the mechanical structure, electronics and software specific to perform the desired task.

The work was very successful, the robot prototype is being used in a real driving range and a Portuguese patent was already submitted.



**Key-words -** autonomous robot, golf, sensors, location, GPS.

# Agradecimentos

Aos responsáveis pelos serviços de Acção Social da Universidade do Minho, por tornarem possível a realização deste projecto.

Aos funcionários da empresa SAR (Soluções de Automação e Robótica), por todo o apoio prestado (Nino Pereira, Ivo Moutinho e Pedro e Silva).

Ao meu orientador, Dr Fernando Ribeiro, pela orientação e ajuda prestada.

Aos meus colegas que estiveram envolvidos no projecto, (André Oliveira, Heloísa Costa, Ricardo Lameiro e Sérgio Oliveira) e colegas de laboratório (Bruno Matos, Cristiano Santos, João Guimarães, Paulo Ricardo e Sílvia Reis) pelo companheirismo durante o tempo que passamos juntos.

Por último mas não menos importante, queria agradecer aos meus pais e irmão o apoio e força que me deram não só ao longo deste trabalho mas também ao longo de todos estes anos como estudante.

# ÍNDICE

# ÍNDICE DE FIGURAS

# ÍNDICE DE TABELAS

# 1. Introdução

Os treinos num jogo de golfe são essenciais. Existem alguns campos onde se podem treinar todas as situações, desde "*bunker*" (areia), "*putt in green*" (última tacada para o buraco) e "*tee-shot*" (tacada de saída) com o nome de "*driving range*". Estes três pontos são extremamente importantes num jogo de golfe. Este tipo de campo de treino apresenta alguns problemas, tais como a necessidade de um número elevado de bolas, tempo de paragem da prática aquando da recolha das mesmas para praticar a tacada de saída, entre outros.

Actualmente a recolha de bolas e sua inserção no sistema dispensador é realizada com intervenção humana e recorrendo ao uso de máquinas com equipamentos atrelados mas sempre conduzidos por humanos. A necessidade de veículos especializados de modo a melhorar a eficácia do sistema impõe-se, não só para proporcionar uma maior rapidez de todo o processo mas também para diminuir os custos de manutenção de todo o sistema.

Os sistemas tradicionais apresentam um desfasamento entre o funcionamento do campo de "*driving range"* e a sua manutenção. É difícil efectuar a recolha de bolas ao mesmo tempo que se encontram jogadores na zona de batimentos – nos casos em que as bolas são recolhidas manualmente é extremamente perigoso para o funcionário andar pelo campo com vários jogadores a fazer batimentos de bolas de golfe (extrema violência).


**Figura 1 Funcionário de um "*Driving-Range*"**

Para solucionar estes problemas surgiu a ideia de se desenvolver uma plataforma móvel autónoma que realize a operação da recolha de bolas de golfe.

## 1.1. Pontos Importantes na Robótica

Os robôs entraram no nosso dia-a-dia, nas mais diversas aplicações. Surgem usualmente a realizar tarefas aborrecidas, perigosas ou maçadoras para o Homem. Aparecem em casa para aspirar, cortar relva, limpar piscinas, tomar conta de crianças, na Indústria manipuladores com funções de paletização e de soldadura, AGV – (*Automated Guied Veichle*) para se movimentar pelo ambiente de trabalho, nos hospitais para distribuir medicação e alimentos pelos doentes, entre muitos outros.

Importa referir aquando da criação de um robô, as três leis da robótica, ou leis de Asimov:

Lei 1 – Um robot não deve fazer mal a um ser humano, ou por omissão, permitir que um ser humano sofra algum dano.

Lei 2 – Um robot deve obedecer às ordens dadas pelos humanos, excepto se essas ordens puserem em causa a primeira lei.

Lei 3 – Um robot deve proteger a sua existência desde que essa protecção não comprometa a satisfação das duas primeiras leis.

Referência Isaac Asimov [17].

## 1.2. Definição de Robô

Devido à imensa diversidade de dispositivos existentes, não existe uma definição que seja universalmente aceite de robô. No entanto, todos os robôs partilham um conjunto de componentes comuns, tais como sistemas de locomoção, sensores e sistemas de processamento. Referência "*Robotics Industries Association"* [20].

Em suma, um robô é um dispositivo mecânico com processamento que pode ser programado para tomar decisões sobre o que fazer com base na informação dos sensores.

## 1.3. Robótica Móvel Autónoma

Todos os robôs móveis e autónomos possuem, um sistema de locomoção, motores, unidade de alimentação, percepção, navegação e actuação, comunicação e uma interface com o utilizador.

O sistema de locomoção permite que o robô se mova. Este pode ser realizado por rodas, pernas, lagartas entre outros colocados das mais diversas posições que oferecem características únicas ao robô influenciando a sua manobrabilidade. Estas características são expressas no modelo cinemático do robô.

Os tipos de motores a utilizar e o modo como fornecer energia aos restantes dispositivos é outro dos pontos fulcrais. Os motores mais usuais são os motores eléctricos e utilizam-se normalmente baterias de chumbo como fonte de energia.

Todos os robôs são construídos com uma determinada missão. Temos desde o simples aspirador que realiza varrimentos pelo chão, até ao robô que vai a Marte recolher amostras do solo. Genericamente pode dizer-se que todas as missões, apesar da sua diversidade, são executadas através de um ciclo composto por:

Percepção → Navegação → Actuação

Na percepção, o robô realiza a leitura do estado dos seus sensores para obter informação acerca do ambiente que o rodeia. Na fase da Navegação, o robô calcula a sua posição e a posição para onde quer ir e ainda qual o caminho a seguir. Na fase da actuação, este realiza o controlo dos motores e dos restantes actuadores.

Outra das componentes importantes é a comunicação, com operadores Humanos, com outros robôs ou entre os vários dispositivos que compõem o robô. É essencial saber qual o estado actual do robô. Saber qual a tarefa que está a correr no momento, qual o estado dos sensores e actuadores. A comunicação pode realizar-se dos mais diversos modos. Podemos usar sons, comportamentos específicos, os mais variados protocolos de comunicação existentes. Referência [6].

Não menos importante, apresenta-se por fim, a interface com o utilizador. Este deve ser o mais simples possível e apresentar a informação essencial para que o utilizador ou parceiro comum do robô possam interagir sem dificuldades.

## 1.4. Objectivos do Trabalho

A participação neste projecto, prendeu-se com a realização de vários desenhos em 3D com vista à criação da estrutura mecânica, dimensionamento do sistema de alimentação, construção de software para aquisição e tratamento dos sinais provenientes dos diversos sensores e controlo de posição, velocidade e navegação do sistema.

# 2. Estado da Arte

## 2.1. Estudar o Golfe

O golfe é oriundo da Escócia do século XV, de uma mistura entre desportos de taco e bola praticados na Holanda e Bélgica.

*"O jogo de Golfe consiste em jogar uma bola desde o ponto de partida até ao buraco, executando uma pancada ou pancadas sucessivas."* Referência 0.

*"A bola de golfe normalmente utilizada é branca e a camada exterior que a cobre tem uma série de covinhas, cerca de 500, para lhe aumentar o efeito de aerodinâmica. Em 1968 foram fixadas, pelas entidades reguladoras do golfe, as especificações técnicas da bola, que constituem o Apêndice III das Regras de Golfe, onde diz que o diâmetro da bola não pode ser inferior a 42,67 mm, nem o seu peso ser superior a 15,93 gr".* Referência [8].

Os campos de treino de golfe que apresentam locais para praticar o "*tee-shot*" são espaços de dimensões consideráveis.

Na sua esmagadora maioria, os sistemas que existem para realizar a recolha de bolas de golfe num campo de treinos, não são autónomos. São sistemas que necessitam de tracção de um qualquer sistema motorizado e intervenção Humana para o conduzir.

**Figura 2 Dispositivo de recolha de bolas**

Os veículos autónomos existentes para ambientes exteriores são bastante limitados, devido às irregularidades do terreno, às diferentes condições ambientais, tais como a variação de luminosidade, a temperatura, a humidade, a própria dimensão do espaço, entre outras.

Para realizar o objectivo a que este trabalho se propõe, existe actualmente em comercialização, um robot que realiza a recolha de bolas de golfe autonomamente – *BallPicker*, *copyright da BelRobotics*. Este robot, foi baseado num cortador de relva eléctrico, possui ultra-sons que lhe possibilitam evitar obstáculos e a sua direcção é controlada alterando a velocidade das rodas (diferencial eléctrico). Este realiza a recolha de bolas, realizando varrimentos ao longo de todo o terreno. Note-se que esta solução não é a mais eficaz, na medida em que, o espaço em questão tem dimensões significativas. É necessário portanto, um sistema que localize quais as zonas do terreno em que existe um maior número de bolas e ainda localmente, para dar ao robô a direcção desejada.

Apresentando algumas das suas características técnicas, temos que, funciona a uma velocidade de 1m/s, reduzindo para 0,3m/s quando o ultra-som detecta um obstáculo. Consegue vencer inclinações de cerca de 30% devido aos dois motores 3 baterias NiCd de 24 volts – 15 Ah. É equipado com uma estação de carga de 30V AC.

Como dimensões, temos um comprimento de 120 cm, largura de 135cm e altura de 50cm. Pesa cerca de 65Kg com o recipiente de bolas vazio.

Para realizar a recolha de bolas, este possui 5 grupos de 4 discos de polietileno acoplados no seu eixo. Tem uma capacidade de recolha de cerca de 200 a 400 bolas por viagem.

Como principal vantagem deste sistema, é de salientar a sua estrutura feita em alumínio, que faz com que este seja leve, optimizando assim o consumo de Energia aumentando a sua autonomia. Referência [25].


**Figura 3** ***BallPicker, copyright da BelRobotics***

# 3. Estudo de Fundo

## 3.1. Controlo PID

### 3.1.1 Teoria PID

O Controlo é uma palavra presente no nosso dia a dia, na medida em que, são inúmeras as situações, ainda que intuitivamente é aplicado. Vejamos que, quando faz frio, tentamos aumentar a temperatura, se o tanque está cheio, fecha-se a torneira, manter a pressão de um gás, a rotação de um motor. Note-se que existe sempre a necessidade de manter um ou mais parâmetros estáveis.

Em todas estas situações é necessário observar o processo em questão, avaliar e tomar a respectiva decisão, ou seja, temos que Medir, Comparar, Decidir e Actuar.

Os dispositivos responsáveis por realizar as funções acima descritas, são o Controlador, o Actuador e o Transmissor. O Controlador gera um sinal de controlo normalizado para o Actuador em função do valor medido da variável que se quer controlar e do seu valor desejado.

Os principais controladores industriais existentes são os Controladores de realimentação (PID – "*Proportional-Integral-Derivative*"), os Autómatos (PLC - "*Programmable Logic Controller*"), os Sistemas de Controlo Distribuído (DCS – "*Distributed Control System*") e o Controlo por computador (PC). Referência [27]

Os Controladores PID são os mais frequentes e podem ser vistos como dispositivos que podem ser controlados a partir de algumas regras práticas. Diz-se por exemplo que, aumenta-se a variável manipulada quando a variável do processo for menor que a referência e diminui-se a variável manipulada quando a variável do processo for maior que a referência. Introduz-se, deste modo o princípio da realimentação, ou seja, manter a variável de processo próxima da referência mesmo na presença de distúrbios e variações nas características do processo. Referência [26].

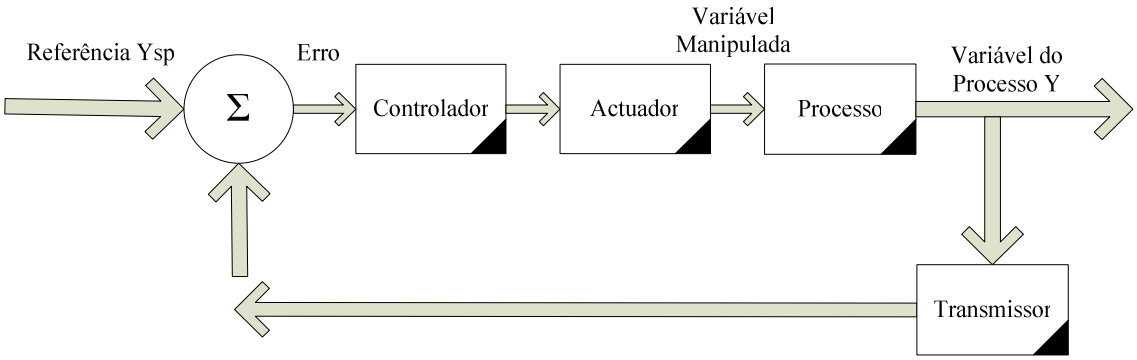


**Figura 4 : Exemplo de Controlador com princípio de Realimentação**

O Algoritmo do Controlador PID, como se pode observar na Equação [1], é a soma de três termos, o Proporcional, o Integrativo e o Derivativo.

**Equação 1**

$$u(t) = K(e(t) + \frac{1}{Ti}\int_0^t e(\tau)d\tau + Td\frac{de(t)}{dt}$$

em que, u(t) é a variável manipulada, *K* o ganho Proporcional, *Ti* a constante de tempo da parte integrativa, *Td* a constante de tempo da parte derivativa e *e(t)* o erro. O erro, pode ser visto como:

**Equação 2**

$$e(t) = Ysp - Y$$

em que, Ysp é o valor desejado, ou seja, o valor referência e Y a variável do processo.

Como principais características da acção de controlo proporcional, temos que, tem grande efeito quando o valor do processo está longe do valor de referência desejado, torna o sistema mais rápido, mas por outro lado, valores elevados originam uma resposta oscilatória e um *overshoot* elevado. Temos ainda que, não permite eliminar o erro em regime permanente. Um controlador unicamente proporcional só pode ser aplicado quando o erro em regime permanente for aceitável dentro de uma determinada gama.

Como principais características da acção de controlo Integral, temos que, permite eliminar o erro em regime permanente que ocorre se usarmos unicamente a acção proporcional mas piora a resposta transitória do sistema.

A acção de controlo Derivativa, aplica-se normalmente quando temos um grande tempo de atraso entre o processo a controlar e o sensor utilizado para a realimentação. Melhora a estabilidade do sistema, reduzindo o *overshoot* e melhora a resposta transitória. Referência [26].

Em suma, na Tabela 1, apresenta-se um resumo das características previamente apresentadas.

**Tabela 1 Resumo PID**

| **Resposta em Malha Fechada** | **Tempo de Subida** | ***Overshoot*** | **Tempo até à estabilização** | **Erro em regime permanente** |
|---|---|---|---|---|
| Kp | Diminui | Aumenta | Pouca Alteração | Diminui |
| Ki | Diminui | Aumenta | Aumenta | Elimina |
| Kd | Pouca Alteração | Diminui | Diminui | Pouca Alteração |

### 3.1.2 Implementação do Controlador PID num computador digital

Uma das grandes dificuldades da implementação de um controlador contínuo num computador digital, é realizar a sua discretização.

Refere-se que, um sinal contínuo *x(t)*, no instante kT (*x(kT)*) é representado por *x(k)*.

Existem vários métodos para realizar a discretização de uma equação. Optou-se por usar o método de *Euler* progressivo para realizar a aproximação da derivada e do integral.

Assim, tendo como base a Equação 1, temos:

- Para o termo integrativo

**Equação 3**

$$\int_0^{kT} e(\tau)d\tau \cong i(k) \approx i(k-1) + Te(k),$$

- Para o termo derivativo

**Equação 4**

$$\frac{de(t)}{dt} = \frac{e(k) - e(k-1)}{T},.$$

Substituindo na Equação 1, temos:

**Equação 5**

$$u(k) = Kp(e(k) + \frac{1}{Ti}i(k) + Td\frac{e(k) - e(k-1)}{T}$$

Note-se que, $Kd = Kp\frac{Td}{T}$ e $Ki = Kp\frac{T}{Ti}$. Fazendo, ainda $s(k) = \frac{1}{T}i(k)$, a Equação 3 e a Equação 5 ficam:

**Equação 6**

$$s(k) = s(k-1) + e(k)$$

**Equação 7**

$$u(k) = Kpe(k) + Kis(k) + Kd(e(k) - e(k-1))$$

Apresenta-se de seguida o algoritmo para implementar a equação obtida.

```
Algoritmo posicional PID(Kp,Ti,Td)
Kd = kp*Td/T;     //Calculo das constants kd e Ki
Ki = Kp*T/Ti;
S=e0=0;           //inicialização de s(k-1) e e(k-1)
Enquanto nao acabar:          //ciclo infinito
      Ler valor desejado e actual (r,y); //ler valor desejado e valor
                                         // medido pelos encoders
      e = r-y;                //cálculo do erro
      s = s+e;                //soma do erro
      u = Kp*e+Ki*s+Kd+(e-e0); //equação controlador
      e0 = e;                 //actualizar o valor anterior do erro
      actualizar u;           //actualizar o novo valor de controlo
Fim
```

Este denomina-se algoritmo posicional, porque ele calcula o valor absoluto do sinal do actuador *u*. Referência [21].

Contrariamente, o algoritmo de velocidade calcula a diferença, ou seja, o valor a incrementar ou decrementar à variável *u*.

O valor a incrementar à variável *u* é proporcional à derivada de *u(t)*. Deste modo, o algoritmo de velocidade fica:

**Equação 8**

$$\frac{du}{dt} = Kp(\frac{de(t)}{dt} + \frac{1}{Ti}e(t) + Td\frac{de(t)^2}{dt})$$

À semelhança do algoritmo anterior, aplicando o método de *euler* à Equação 8, temos:

**Equação 9**

$$\frac{de(k)^2}{dt} = \frac{e(k) - 2e(k-1) + e(k-2)}{T^2}$$

A Equação 8, fica então aproximada a:

**Equação 10**

$$\frac{u(k) - u(k-1)}{T} = Kp(\frac{e(k) - e(k-1)}{T} + \frac{1}{Ti}e(k) + Td\frac{e(k) - 2e(k-1) + e(k-2)}{T^2}$$

Fazendo

$$\Delta u(k) = Kp((e(k) - e(k-1)) + \frac{T}{Ti}e(k) + \frac{T}{Ti}e(k) + \frac{Td}{T}(e(k) - 2e(k-1) + e(k-2)))$$
$$= Kp((1 + \frac{T}{Ti} + \frac{Td}{T})e(k) - (1 + 2\frac{Td}{T})e(k-1) + \frac{Td}{T}e(k-2))$$

Apresenta-se de seguida o algoritmo para implementar a equação obtida.

```
Algoritmo de velocidade PID(Kp,Ti,Td)
K1 = kp*(1+T/Ti+Td/T);  //Calculo das constants kd e Ki
K2 = -Kp*(1+2*Td/T);
K3 = Kp*Td/T;
u=e0=e1=0;        //inicialização de e(k-1) e e(k-2)
Enquanto nao acabar:          //ciclo infinito
      Ler valor desejado e actual (r,y); //ler valor desejado e valor
                                         // medido pelos encoders
      e = r-y;                //cálculo do erro
      u = u + K1*e+K2*e0+K3*e1; //equação controlador
      e1 = e0;                //actualizar o valor anterior do erro
      e0 = e;
      actualizar u;           //actualizar o novo valor de controlo
Fim
```

Um aspecto bastante importante a ter em conta é o tempo do ciclo de controlo. Os ciclos infinitos apresentados em cima, tem que correr exactamente e cada *T* segundos.

Existem vários métodos para realizar esta sincronização, tais como adicionando *polling*, sinais provenientes de interrupções externas, adicionando um *real-time clock* entre outros. Referência [21].

### 3.1.3 Métodos Sintonia

Na maioria das aplicações, a função de transferência do sistema a controlar mal se conhece, pode ainda apresentar não linearidades e variação ao longo do tempo. Nestes casos, para sintonizar os parâmetros, usam-se métodos experimentais, tais como o método de *Ziegler-Nichols*, *Cohen-Coon*, *Chien-Hrones-Reswick*. Estes, somente exigem a execução de testes simples em malha aberta ou em malha fechada, sem necessidade de realizar a modelização do sistema. Servem para obter uma estimativa inicial, necessitando posteriormente de um ajuste fino. Referência [27].

Por exemplo para o método de *Ziegler-Nichols* de malha aberta, temos:

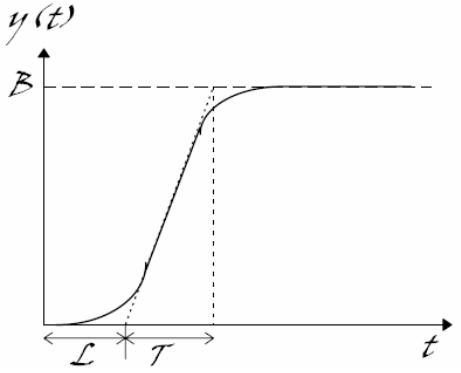


**Figura 5 *Ziegler-Nichols* manha aberta**

$$R = \frac{B}{T} \qquad Kc = \frac{1,2}{RL} = \frac{1,2T}{BL}$$

**Tabela 2 Regras de Ziegler Nichols malha aberta**

| | **Kc** | **τi** | **τd** |
|---|---|---|---|
| **P** | $1/RL$ | - | - |
| **PI** | $0,9T/L$ | $L/0,3$ | - |
| **PID** | $1,2T/L$ | $2L$ | $0,5L$ |

$$Kp = Kc \qquad Ki = \frac{Kc}{\mathrm{T}i} \qquad Kd = Kc \times Td$$

Para o método de *Ziegler-Nichols* de malha fechada, usando somente um ganho proporcional, (*Ku*) ajusta-se até o sistema a apresentar oscilações constantes.

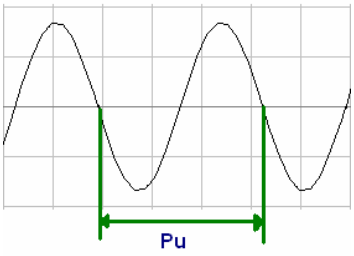


**Figura 6 Ziegler-Nichols manha fechada**

De seguida, lê-se o período de oscilação (*Pu*).

**Tabela 3 Regras de Ziegler Nichols malha fechada**

| | **Kc** | **τi** | **τd** |
|---|---|---|---|
| **P** | $Ku/2$ | - | - |
| **PI** | $Ku/2{,}2$ | $Pu/1{,}2$ | - |
| **PID** | $Ku/1{,}7$ | $Pu/2$ | $Pu/8$ |

Temos então,

$$Kp = Kc\,; \qquad Ki = Kp/Ti\,; \qquad Kd = Kp \times Td\,;$$

Deste modo, obtêm-se os valores a aplicar no controlador do sistema. Referência [27].

## 3.2. GPS

O nome GPS (*Global Positioning System*) surge associado a um dispositivo capaz de realizar a localização de um ponto em qualquer parte do mundo.

Teve início num projecto do *Department of Defense* (DOD) dos Estados Unidos, onde o objectivo era criar um sistema para estimar uma posição de um ponto à superfície da Terra, baseado em satélites.

### 3.2.1 Introdução

Trata-se de um sistema que usa ondas de rádio enviadas por satélites para determinar a posição de um receptor.

Como principais características, temos:

- ➔ Coordenadas de posição em qualquer do mundo;
- ➔ Fisicamente é constituído por uma rede de satélites, a uma distância de 20180 km da superfície terrestre, os quais emitem sinais, demorando cada satélite 12 h a dar uma volta completa à Terra;
- ➔ Rede de 5 estações de rastreio, 3 antenas terrestres e 1 central de controlo (MCS), em Colorado *Springs*, *Schriever AFB*, onde a órbita de cada satélite é constantemente monitorizada, podendo cada satélite receber instruções para corrigir a sua órbita, por causa das atracções gravitacionais do Sol e da Lua, bem como do efeito da pressão da radiação solar;
- ➔ A precisão do tempo é essencial no funcionamento do GPS. Um erro de um micro segundo ($10^{-6}$ segundos) desde a transmissão até à recepção do sinal, resulta num erro de 300 metros. Referência [1].

Para localizar um ponto no espaço, é necessário conhecer a posição exacta dos satélites. A informação enviada divide-se em duas parte, as suas efemérides[1] e o almanaque[2].

Os sinais enviados pelos satélites deslocam-se à velocidade da luz c ≈300,0×10^6 (m/s). A diferença de tempos entre o instante em que o sinal é emitido pelo satélite e o instante em que o sinal é recebido pelo receptor, permite determinar a distância a que o receptor se encontra do satélite, fazendo, $v = \Delta x / \Delta t$ em que v é a velocidade, $\Delta x$ a distância e Δt o tempo. Referência [1].

### 3.2.2 Erros

Os sinais recebidos pelos receptores GPS estão sujeitos a inúmeros erros. Resumidamente, referem-se as imprecisões dos relógios, dos satélites que são de alta precisão e dos receptores, que se traduzem em erro no cálculo da distância ao satélite. A aproximação da velocidade de transmissão dos dados à da luz, o que realmente não acontece ao atravessar a atmosfera terrestre. Note-se ainda, a existência de superfícies reflectoras perto dos receptores que provocam degradação do sinal. O facto da Terra não ser redonda, a posição dos satélites aquando da medição, entre outros. Referência [3].

[1] Efemérides – Conjunto de parâmetros que permitem calcular a posição do satélite no espaço num dado instante t.

[2] Almanaque – características aproximadas das órbitas dos restantes satélites da constelação.

### 3.2.3 Método de Triangulação

Apresenta-se na Figura 7 o método de Triangulação utilizado para realizar a localização de um qualquer ponto.

Usando somente um satélite, consegue-se saber qual a distância a que o ponto se encontra, mas não o conseguimos localizar. Seja A um satélite e P o ponto que se pretende localizar. Sabe-se que o ponto P está à distância d do ponto A, mas este pode estar em qualquer parte da circunferência, como apresentado. Adicionando outro satélite B, obtêm-se dois pontos comuns a ambos. Adiciona-se portanto um terceiro satélite C, para se obter um ponto único comum aos três satélites.

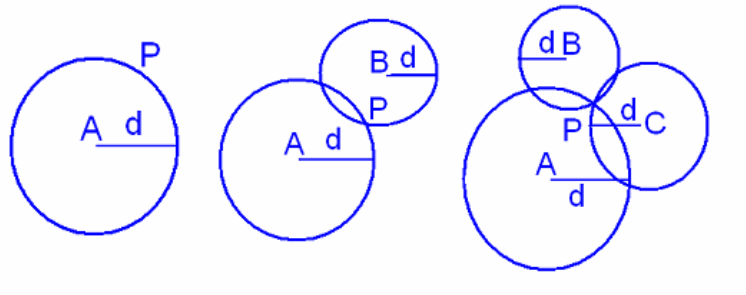


**Figura 7 Método de Triangulação por intercepção de esferas**

Recorre-se ainda a um quarto satélite, cujo objectivo é sintonizar os relógios atómicos bastante precisos que estão nos satélites, com relógios de quartzo menos precisos existentes nos receptores. Referência[1].

É assim determinada a posição tridimensional de um receptor, cujas coordenadas são a latitude, longitude e altitude.

A Terra não é uma esfera, porque é achatada nos pólos e acidentada à superfície, mas considera-se como tal quando pretendemos determinar as coordenadas de um ponto qualquer à sua superfície. Divide-se em dois hemisférios, o Norte e o Sul pela linha imaginária do equador, por planos paralelos ao plano equatorial com menor diâmetro e por meridianos, que são linhas perpendiculares ao equador onde se destaca o meridiano de Greenwich como meridiano de origem para contagem de longitude.

**Figura 8 Linha do Equador**

**Figura 9 Meridiano de Greenwich**

O valor do ângulo medido entre o paralelo que passa pelo local desejado e o equador tendo como origem o centro da Terra, denomina-se **latitude.** A Latitude varia de 0º a 90º com origem no equador, considerando latitude positiva para Norte e negativa para Sul. Referência [3]

A **longitude** é o valor do ângulo medido entre o meridiano que passa pelo local desejado e o meridiano de Greenwich com origem no centro da Terra. A Longitude varia de 0º a 180º, considerando negativa para oeste e positiva para leste a partir do meridiano de Greenwich. Referência [3].

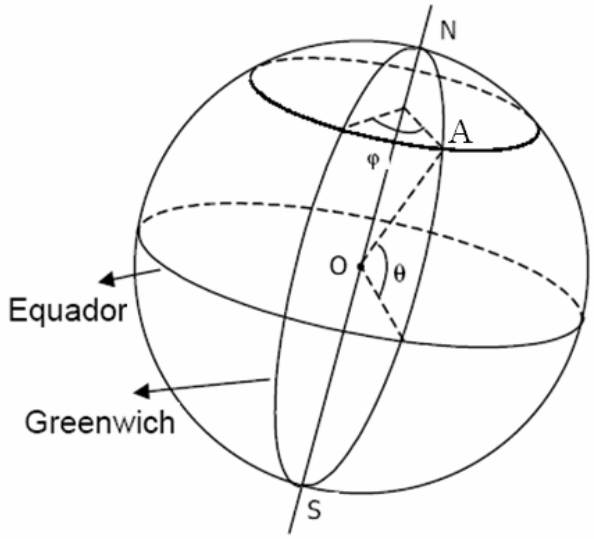


**Figura 10 Latitude / Longitude**

Onde **θ** é a latitude e **φ** a longitude do ponto A.

### 3.2.4 Coordenadas geográficas / coordenadas cartesianas

Considerando o sistema referencial apresentado na Figura 11, colocando o ponto O no centro da Terra, considerando o eixo OZ positivo uma linha que intercepta o pólo Norte, o plano OXY que paralelamente intercepta a linha do equador, o eixo OX positivo uma linha perpendicular ao meridiano de Greenwich e o eixo OY positivo que perpendicularmente intercepta a linha do meridiano de Longitude 90º. Referência [28].

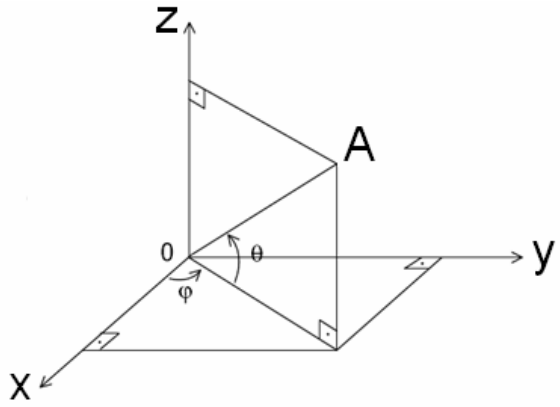


**Figura 11 Coordenadas cartesianas**

Considerando o ponto A colocado à superfície da Terra, temos que **θ** é a latitude e **φ** a longitude do ponto A. Considere-se ainda a distância OA o valor do raio da Terra *r*, tem-se:

$$x = r \cos \theta \cos \varphi$$
$$y = r \cos \theta \operatorname{sen} \varphi$$
$$z = r \operatorname{sen} \theta$$

### 3.2.5 Distância entre dois pontos numa superfície esférica

Quanto maior for o raio de uma circunferência, mais esta se aproxima de um plano, ou seja, se a distância a calcular for muito menor que o valor do raio, pode-se usar o cálculo de distância entre dois pontos num plano (teorema de Pitágoras). Referência [1].

No entanto para realizar o cálculo entre dois pontos numa superfície esférica, pode usar-se:

**Equação 11**

$$d = \sqrt{\Delta X^2 + \Delta Y^2 + \Delta Z^2}$$

### 3.2.6 Posicionamento

#### a) Absoluto

Este tipo de posicionamento somente utiliza uma antena receptora e é baseado no tempo de propagação do sinal desde o satélite até ao receptor.

#### b) Relativo a Bases

Neste tipo de posicionamento, são necessários pelo menos dois receptores. Coloca-se um dos receptores numa posição de coordenadas conhecidas e a cada leitura é calculado o erro e enviado, geralmente por rádio frequência, para o receptor cuja posição queremos saber. Como ambos os receptores recebem informação dos mesmos satélites, o erro recebido por ambos é semelhante, conseguindo-se deste modo a sua atenuação. Referência [1].

### 3.2.7 WAAS, EGNOS e MSAS

Existem actualmente várias redes de estações de referência, de posição conhecida, cuja função é enviar para os receptores GPS, que estejam no seu alcance, parâmetros para atenuar o erro, tendo por base o sistema de posicionamento "Relativo a Bases". Estas usam satélites geostacionários para enviar os parâmetros de correcção. Cada receptor fica ligado aos quatro satélites e à estação mais próxima da rede em questão para fazer o posicionamento.

Usando receptores GPS que permitam troca de informação com este sistema, consegue-se atenuar o erro para valores próximos dos três metros.

Existem vários sistemas, grupos conjuntos de estações de referência, entre os quais se destacam o Americano, WASS (*Wide Area Augmentation System*), o Europeu EGNOS (*European Geostationary Navigation Overlay Service*) e o utilizado pelo Japão e outros países asiáticos, MSAS (*Multi-Functional Satellite Augmentation System*). Referência [2].

### 3.2.8 Norte Geográfico / Norte Magnético

Existem dois tipos de norte, o norte Geográfico (Pólo norte, em torno do qual gira a Terra) e o norte magnético (o norte indicado pelas bússolas).

O norte indicado pelas bússolas, é consequência do ferro existente no núcleo da terra, que faz com que este não seja coincidente com o norte geográfico.

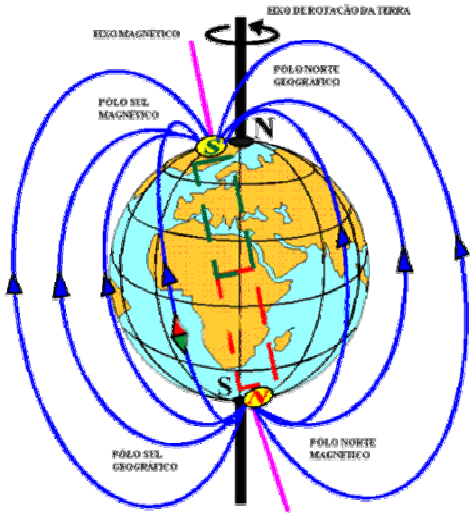


**Figura 12 Norte Magnético e Norte Geográfico**

Este fenómeno denomina-se de Declinação Magnética e esta varia consoante o local do mundo. Em certas zonas do Canadá ultrapassa os 40 graus, mas, por exemplo, na Escandinávia ela é desprezável. Em Portugal, a declinação é de cerca de 7º. Referência [29].

A título de curiosidade, apresenta-se na Figura 13 um mapa com a variação do campo magnético na Terra. Referência [30].

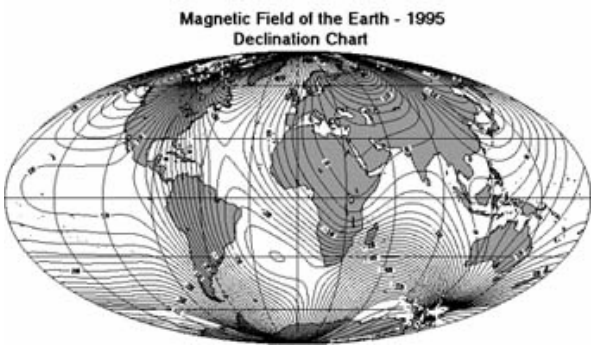


**Figura 13 Campo magnético da terra em 1995**

### 3.2.9 Interface GPS

A interface com os receptores GPS é feita de dois modos. No modo binário ou usando tramas no formato NMEA[3]. As tramas usadas no formato NMEA, têm o formato ASCII[4] e apresentam uma estrutura muito própria.

Apresenta-se de seguida um exemplo de uma trama NMEA em ASCII.

$GPRMC,225446,A,4916.45,N,12311.12,W,000.5,054.7,191194,020.3,E*68

Note-se que todas as tramas começam por $, seguindo-se a identificação do tipo de receptor (GP = GPS), do tipo de trama (por exemplo, RMC - *Recommended minimum specific* GPS/TRANSIT data) e da informação que se apresenta separada por vírgulas. Apresentam-se na Tabela 4, alguns dos tipos de tramas NMEA. Referência [4].

**Tabela 4 NMEA Output messages**

| Trama | Descrição |
|---|---|
| GGA | Time, position and fix type data. |
| GLL | Latitude, longitude, UTC time of position fix and status. |
| GSA | GPS receiver operating mode, satellites used in the position solution, and DOP values. |
| GSV | The number of GPS satellites in view satellite ID numbers, elevation, azimuth, and SNR values. |
| MSS | Signal-to-noise ratio, signal strength, frequency, and bit rate from a radio-beacon receiver. |
| RMC | Time, date, position, course and speed data. |
| VTG | Course and speed information relative to the ground. |
| ZDA | PPS timing message (synchronized to PPS). |
| 150 | OK to send message. |

[3] NMEA - (National Marine Electronics Association) – entidade do DOD que desenvolveu o protocolo de linguagem do sistema de GPS que lhe dá nome.

[4] ASCII – esquema de codificação de caracteres padrão, utilizado para dados baseados em texto. Recorre-se a utilização de números específicos (de 7 ou 8 bits), para representar 128 ou 256 caracteres

As mensagens no modo binário são compostas por 5 campos obrigatórios.

O primeiro é a mensagem de início, (A0A2), seguida do tamanho da informação útil presente na trama (2 bytes). Segue-se a informação ($2^{10} - 1$ =1023 bits), o checksum (2 bytes) e a mensagem de fim (B0B3). Referência [5].

Apresenta-se de seguida um exemplo:

A0 A2 00 5B— Sequência de início e tamanho da informação.

29 00 00 02 04 04 E8 1D 97 A7 62 07 D4 02 06 11 36 61 DA 1A 80 01 58 16 47 03 DF B7 55 48 8F FF FF FA C8 00 00 04 C6 15 00 00 00 00 00 00 00 00 00 00 00 00 00 BB 00 00 01 38 00 00 00 00 00 00 6B 0A F8 61 00 00 00 00 00 1C 13 14 00 00 00 00 00 00 00 00 00 00 00 00 08 05 00— informação

11 03 B0 B3— *Checksum* e sequência de fim.

## 3.3. Barramento I2C

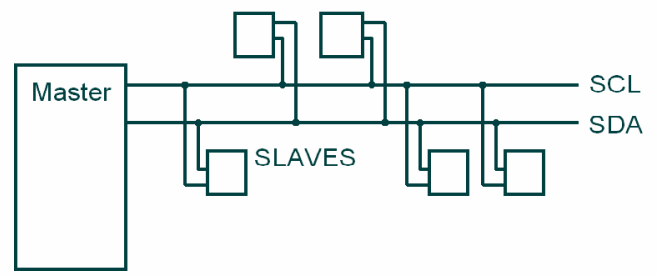


**Figura 14 Barramento i2c**

É um protocolo de comunicação série criado pela Philips, "*Inter-Integrade Circuit*". O objectivo inicial era a comunicação entre integrados, daí que este protocolo se encontre facilmente em muitos componentes. A comunicação é feita por duas linhas TTL e permite o endereçamento dos diversos dispositivos colocados ao longo de um barramento.

São utilizadas duas linhas de comunicação, serial data (SDA) e serial clock (SCL). Nestas linhas temos informações digitais, ou seja, operam dentro dos limites de 0 a 5 volts. Para que o sistema funcione correctamente, temos que obedecer aos requisitos do protocolo, ou seja, o sinal de start (início da transmissão), o sinal de stop (fim da transmissão), entre outros.

Pela sua simplicidade, tornou-se um standard em comunicação, sendo usado nos mais diversos dispositivos. Define-se de seguida o funcionamento do protocolo.

A linha de *clock* fornece a cadência para que a transmissão série seja entendida. Por transmissão série, entende-se que a informação é enviada uma após a outra através da mesma linha digital. Deste modo, é a linha de *clock* quem identifica quando um dado (um bit) pode ser considerado válido - esta situação é sempre definida quando a linha de *clock* está no nível lógico alto. Assim, sempre que a linha de *clock* SCL estiver no nível lógico alto, sabemos que a linha SDA possui um dado válido.

Para dar início a uma transmissão, temos que realizar o sinal de *start*. A linha SDA transita do nível lógico alto para o nível lógico baixo (transição). A linha SCL transita de nível baixo para alto (transição).

```
i2c_start()
      SCL(0)
      SDA(1)
      SCL(1)
      SDA(0)
```

Os diversos dispositivos colocados no barramento, são identificados por um endereço. Para tal, são reservados 7 ou 10 bits a seguir ao sinal de *start*. Somente ao dispositivo endereçado, é que serão dirigidos os dados.

Para terminar uma transmissão, temos que realizar o sinal de stop. Ambas as linhas, SDA e SCL devem passar de nível baixo para alto (transição).

```
i2c_stop()
       SCL(0)
       SDA(0)
       SCL(1)
       SDA(1)
```

Nos protocolos de transferência de dados, podemos distinguir, dois tipos de dispositivos, o principal, normalmente um sistema de processamento, encarregue de gerir o sistema, designado por *Master*, e os dispositivos secundários, que recebem ordens, do dispositivo principal, denominados por *Slaves*.

Para que não existam erros na transferência, as condições devem ser reconhecidas, (*acknowledge*). Deste modo, após a condição de *start*, e o endereçamento estarem concluídos, o *slave* seleccionado fornece um sinal de reconhecimento (*ack*).

Quando o barramento i2c se encontra desocupado, ambas as linhas SDA e SCL apresentam-se no nível lógico alto.

Apresentam-se de seguida, algumas das características do protocolo i2c:

Não fixa a velocidade de transmissão, sendo esta determinada pelo circuito Master, ao realizar a transmissão do sinal de SCL;

Possibilidade de inclusão ou exclusão de dispositivos no barramento sem afectá-lo, ou outros dispositivos conectados a este;

Qualquer dispositivo conectado pode operar como transmissor ou receptor;

Elevado número de dispositivos existentes no mercado, que utilizam este protocolo de comunicação, existindo até conversores de USB, SÉRIE entre outros, para i2c. Referência [32] e [31].

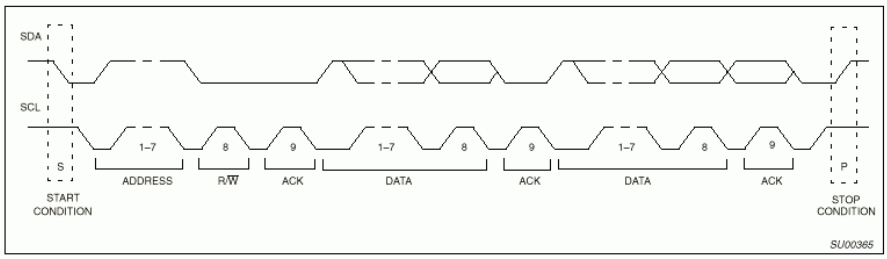


**Figura 15 Exemplo comunicação i2c**

Apresenta-se, o procedimento de comunicação do protocolo I2C. Basicamente temos 6 itens para análise:

- O dispositivo *master* ajusta a condição inicial;
- O dispositivo *master* envia 7 bis de endereçamento;
- O dispositivo *master* envia o 8º bit, RW/;
- O dispositivo *slave* envia o sinal de ACK (*Acknowledge*);
- O dispositivo *master* (ou *slave*) envia pacotes de 8 bits de dados, sempre seguidos de um sinal ACK enviado pelo dispositivo *slave* (ou *master*) confirmando a recepção;
- O dispositivo *master* encerra a comunicação;

# 4. Criação da Estrutura Mecânica

Foi realizado um estudo inicial, de forma a criar uma estrutura mecânica capaz de realizar a tarefa pretendida para este robot. Primeiro, foram avaliados os sistemas que existem para a recolha de bolas no mundo do golfe apresentados na Figura 16.

**Figura 16 Sistemas de recolha de bolas**

Posteriormente, foram desenhadas em CAD 3D algumas das ideias que surgiram, de onde se destacam as apresentadas na Figura 17.

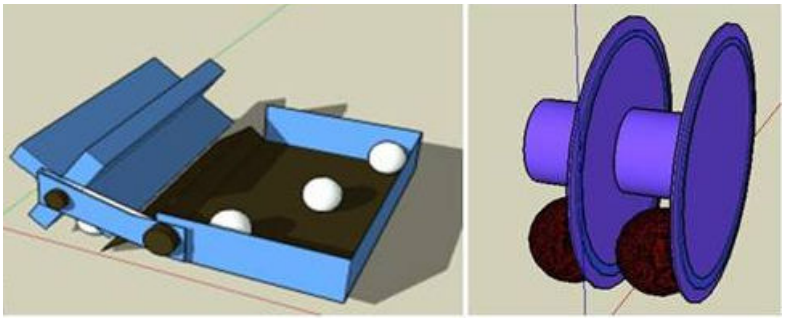

**Figura 17 Desenhos 3D sistema de recolha**

No primeiro caso, é usado um sistema que empurra as bolas para um recipiente, solução que após alguma discussão foi descartada devido à pouca eficiência. No segundo caso, à semelhança dos sistemas já existentes à venda, as bolas ficam entaladas e são posteriormente elevadas e despejadas para um cesto. Esta foi a solução adoptada visto tratar-se de uma solução bastante testada sendo mesmo a mais usada para realizar a recolha das bolas num campo de golfe.

De seguida era necessária a criação de uma estrutura mecânica capaz de puxar todo o sistema. Foram então colocadas várias propostas em cima da mesa, propostas estas que foram desenhadas, e que são apresentadas de seguida.

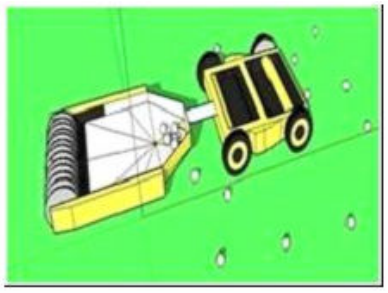

**Figura 18 Modelo 1**

Na Figura 18, é apresentada a primeira hipótese, como sendo um veículo de quatro rodas que puxa um atrelado. Esta foi rejeitada, por não existir necessidade de ter quatro rodas e à sua complexidade e dificuldade de controlo, no que diz respeito à realização de manobras.

**Figura 19 Modelo 2**

Na Figura 19, é apresentada a segunda hipótese, constituída por um sistema que integra o rolo que realiza a recolha das bolas com duas rodas livres e que servem de apoio a duas outras rodas com tracção. Este sistema possui diferencial eléctrico e foi também rejeitado, porque como a estrutura possui dimensões significativas e este não era capaz de virar exercendo forças excessivas nos motores.

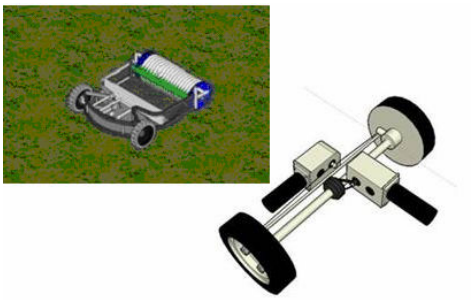

**Figura 20 Modelo 3**

Alterou-se ligeiramente a solução anterior, apresentada na Figura 20, colocando direcção nas rodas da frente juntamente com a tracção e usando um

diferencial mecânico. Esta foi também rejeitada, devido ao tamanho e peso excessivo do sistema, que ao curvar arrastava o rolo.

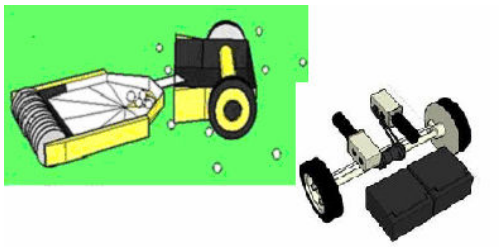

**Figura 21 Modelo 4**

Adoptou-se por fim, um sistema semelhante aos utilizados nos reboques, e atrelados apresentado na Figura 21, permitindo assim que este curve sem qualquer problema usando um diferencial eléctrico.

Como o protótipo foi evoluindo, a solução actual não é a desejada.

Deste modo, a solução desejada, passa por acoplar directamente os motores às rodas, evitando assim engrenagens e folgas, criando também um diferencial eléctrico, retirando peso excessivo à estrutura, deixando ainda espaço para aumentar a autonomia do sistema, aumentando o número de baterias, como apresentado na Figura 22.

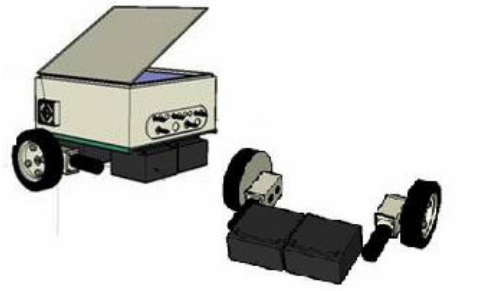

**Figura 22 Modelo 5**

Como se trata de uma estrutura para exteriores, esta tem de ser capaz de resistir ao calor e/ou humidade adversas, bem como a todas as diferentes condições ambientais. Dotou-se então o sistema com uma caixa estanque e dimensionou-se um sistema de ventilação com filtros para proteger o sistema.

Para realizar a recolha de bolas usou-se o dispositivo apresentado na Figura 23 V-Collector. Como principais características, temos:

- Largura(exterior) -100 cm
- Profundidade – 68 cm
- Altura (cestos) – 31cm
- Peso – 21Kg

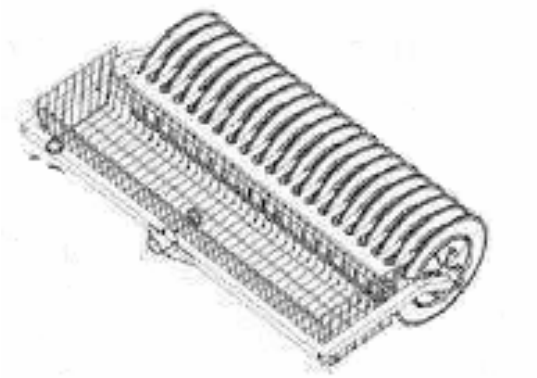

**Figura 23 V-Collector**

Apresenta-se na Figura 24 um conjunto de imagens, para poder observar e analisar o estado do protótipo actual.

**Figura 24 Robô Golfinho**

# 5. Descrição do Hardware Utilizado

Depois de construída uma estrutura mecânica, capaz de realizar a tarefa desejada, é necessário seleccionar um conjunto de sensores e actuadores, de forma a conseguir realizar a tarefa pretendida.

Apresenta-se na Figura 25, o hardware para realizar a tarefa pretendida, assim como o modo como este está interligado.

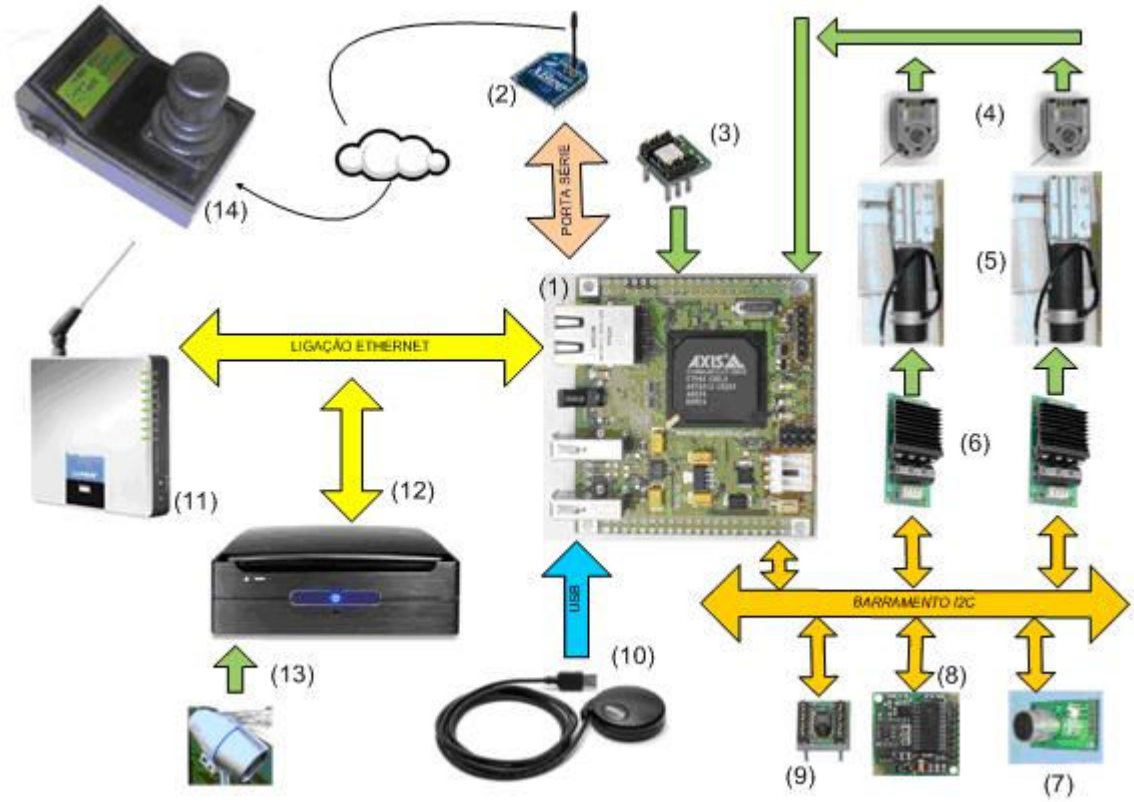


**Figura 25 Diagrama geral de todo o sistema**

Neste esquema é possível ver o processador contínuo *FoxBoard* (1), um módulo de *Xbee* (2), um acelerómetro (3), os *encoders* (4), os motores (5), as placas de controlo dos motores *MD03* (6), um ultra-som (7), uma bússola electrónica (8), um sensor de humidade (9), um receptor GPS (10), um router *wireless* (11), uma unidade de processamento de imagem (12), uma câmara (13) e um *Joystick* (14).

Como principais ligações entre os vários dispositivos, importa referir o barramento *i2c* entre as placas de controlo dos motores, bússola ultra-som e sensor de humidade e uma ligação TCPIP entre o processador contínuo *FoxBoard*, o router wireless e a unidade de processamento de imagem.

## 5.1. Controlador

Com vista a dotar o robô de capacidade de percepção do ambiente que o rodeia, foi necessário escolher uma unidade de processamento. Trata-se da construção de um sistema que ofereça robustez, permita a fácil expansão, uma vez que o projecto se encontra numa fase de protótipo e ofereça fiabilidade na execução de um algoritmo com alguma complexidade. É necessário monitorizar continuamente o estado de um número considerável de periféricos, realizar comunicação com outras unidades de processamento e realizar o controlo dos actuadores necessários.

### 5.1.1 FOXBOARD

A escolha do controlador recaiu numa FOXBOARD cuja descrição das principais características se apresenta de seguida.

**Figura 26 Processador Contínuo FoxBoard**

Caracteriza-se por ser um dispositivo que possui um sistema operativo Linux embebido, com um CPU ETRAX 100LX (100 MIPS) 32bit com arquitectura RISC desenvolvido por AXIS [33], com tamanho reduzido, consumos energéticos baixos, especialmente desenhado para permitir uma fácil integração nas mais diversas aplicações.

Possui 64Mb de memória RAM de 32MB de memória flash no seu modelo com melhores requisitos, tendo o modelo básico 16MB de memória RAM e 4MB de memória flash.

Como interfaces, possui uma porta *Ethernet* 10/100Mb, duas portas USB [5] 1.1, dois *slots* 20 pinos livres com 48 linhas I/O, barramento I2C, SPI[6], portas série e paralelas, mas no entanto não estão todos disponíveis em simultâneo.

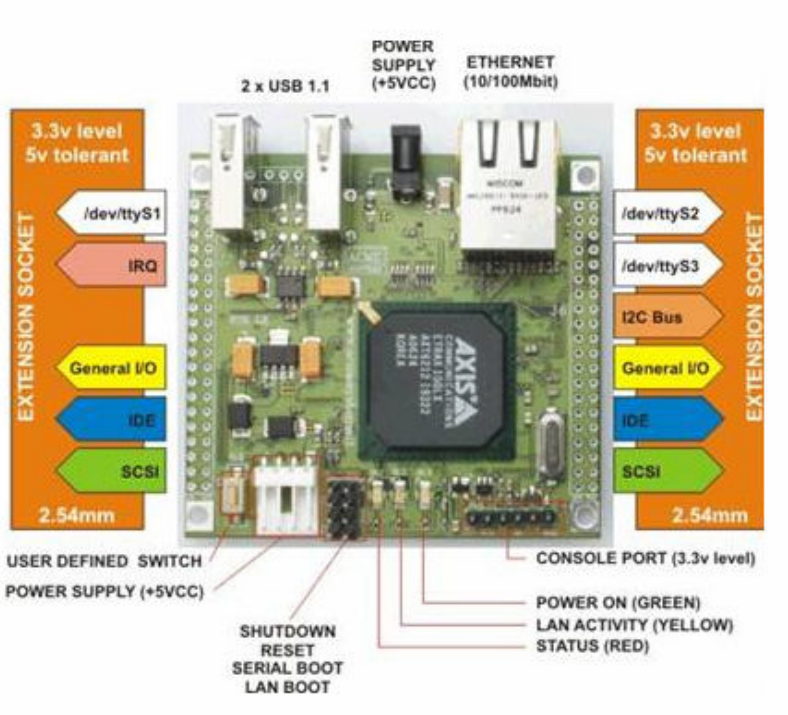


**Figura 27 Fox Board Periféricos**

Possui baixos consumos de energia, 5V/1WATT.

Contém um sistema operativo LINUX com KERNEL 2.4.x ou 2.6.x.

É compatível com um número significativo de linguagens de programação, C, C++, JAVA, PHP, entre outras, bem como com o software *Open Source Linux* disponível.

Pode funcionar como servidor WEB, FTP, TELNET e SSH entre outros.

Tem um tempo de arranque inferior a 10 segundos e todo o seu *firmware* pode ser actualizado através de sessões LAN, WEB ou FTP.

Possui ainda um extenso suporte técnico e vem com exemplos de inúmeras aplicações que demonstram as reais capacidades da FOXBOARD. Referência [9].

[5] USB- Universal Serial Bus é um tipo de conexão Plug and Play que permite a conexão de periféricos sem a necessidade de desligar o computador.

[6] SPI – Serial Port Interface é uma barramento série que funciona de modo síncrono.

### a) Início

Para usarmos a *FoxBoard*, é necessário fornecer-lhe uma alimentação de 5V e criar uma rede para aceder ao *WebServer* interno, transferir ficheiros através de uma sessão FTP ou ter acesso à consola através de uma sessão de TELNET.

O seu endereço IP por defeito é 192.168.0.90 e a mascara de rede é 255.255.0.0.

Para abrir o servidor Web, executa-se no browser:

http://192.168.0.90


**Figura 28 Fox Board WEB PAGE**

Para dar início a uma sessão telnet, basta executar o comando:

telnet 192.168.0.90

e usar o **login**: root e a **password**: pass

Podemos ainda usar o programa *HyperTerminal* do Windows ou usar uma aplicação chamada de *PuTTY* disponível em http://www.acmesystems.it/?id=53.

### b) Como compilar

Para compilar um ficheiro em C, existem dois modos de o fazer. Pode-se utilizar o *WebCompiler* 0, ou instalar o SDK0 num sistema operativo Linux.

Para utilizar o *WebCompiler*, é necessário realizar o upload do código para http://www.acmesystems.it/?id=200 e fazer o download do ficheiro executável que

posteriormente é necessário copiar para a *FoxBoard* através de um dos métodos acima descritos.

Depois de instalado o SDK num sistema operativo Linux, abre-se uma consola e executa-se o seguinte comando na respectiva directoria:

```
/home/fox/devboard-R_01# . init_env
```

De seguida, usando um ficheiro MAKEFILE, disponível para *download* em http://www.acmesystems.it/?id=59, compila-se o ficheiro editado. Posteriormente, realiza-se também a cópia do ficheiro executável através dos processos já descritos.

### c) Funções Básicas

O método utilizado para realizar a interacção com os pinos, mudar o estado e realizar a leitura, foi o *File descriptor method*.

Os vários pinos disponíveis são divididos por portos (A, B e G)- Por exemplo, para verificar qual o estado de um pino, é necessário saber a que porto corresponde esse pino, consultando o *pinout* da FOXBOARD [34].

Antes de realizar qualquer aplicação que envolva cada um dos portos, é necessário abrir um ficheiro, */dev/gpioa* para o porto A, */dev/gpiob* para o porto B e */dev/gpiog* para o porto G.

Por exemplo para o porto G, temos:

```
fd = open("/dev/gpiog", O_RDWR);
```

No fim da execução, é necessário fechar o ficheiro.

```
close(fd);
```

Por exemplo, para colocar o bit especificado por iomask no estado "1", faz-se:

```
ioctl(fd,_IO(ETRAXGPIO_IOCTYPE,IO_SETBITS),iomask);
```

Para colocar o pino no estado "0", faz-se:

```
ioctl(fd,_IO(ETRAXGPIO_IOCTYPE,IO_CLRBITS),iomask);
```

Por exemplo para activar as linhas 3, 4 e 5 do porto G, faz-se:

```
iomask = 1<<5 | 1<<4 | 1<<3;
```

Neste caso, iomask = 00000000 00000000 00000000 00111000

Para realizar a leitura do estado de um pino, faz-se:

```
Value = ioctl(fd,_IO(ETRAXGPIO_IOCTYPE, IO_READBITS));
value = value&iomask;
```

O estado do pino é lido pela variável *value* e o pino é definido pela mascara *iomask*. Referência [35].

## 5.2. Sensores

Uma característica a ter em atenção num robô móvel é o conjunto de sensores a utilizar. Note-se que é muito importante o robô ter conhecimento correcto do ambiente que o rodeia e da evolução deste, uma vez que anda por ambientes não estruturados.

### 5.2.1 Humidade

**Objectivo:**

Analisar o valor de humidade para verificar se o robot pode operar normalmente sem danificar os diversos componentes.

**Princípio de Funcionamento:**

Normalmente, existe humidade no ambiente que nos rodeia. O número de moléculas de água no ar variam substancialmente, ou seja, o ar pode ser seco como no deserto, ou húmido como nos trópicos. Existe um limite superior de quantidade de humidade que o ar pode suportar a uma dada temperatura. Se por alguma razão, o valor de humidade ultrapassar esse valor, ocorre o fenómeno chamado condensação. Denomina-se este limite como *dew-point*, a temperatura para a qual uma determinada quantidade de humidade presente no ar começa a condensar.

**Esquema Eléctrico:**

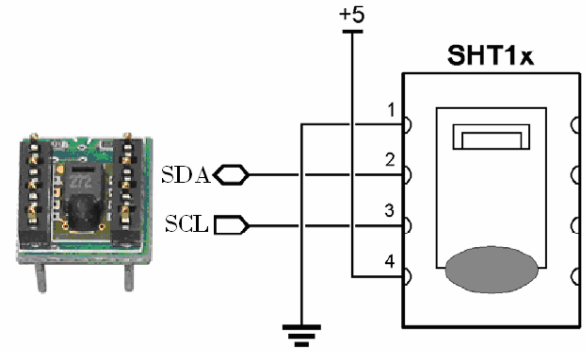


**Figura 29 Sensor de Humidade**

**Características técnicas:**

O sensor utilizado, *STH11* possui polímeros capacitivos, capazes de calcular a humidade relativa e sensores de temperatura acoplados a um ADC [7]de 14 bits que fornece informação à interface i2c.

[7] ADC – Conversor Analógico Digital, fornece uma aplicação digital de um sinal analógico.

**Como é controlado:**

Este sensor é controlado através do barramento i2c ligado a *FoxBoard*.

**Software realizado:**

Foi realizada uma rotina para realizar o cálculo do valor da humidade `void Calcula_Humidade(void)`. Esta é chamada na rotina que percorre o barramento *i2c*.

```
Calcula_Humidade
        medir a humidade
        medir a temperature
        Se (erro de leitura > o)
                Reset da comunicação
        Senão
                Calcular humidade e temperatura
                Calcular o dew point
```

Para realizar a medição da humidade e a temperatura, é usada a rotina `s_measure(int *p_value, unsigned char *p_checksum, unsigned char mode)`.

```
s_measure()
        inicio da transmissão
        escolha do valor a medir (Humidade ou Temparatura)
        esperar 55ms para realizar a medição
        ler valor
```

Para dar início à transmissão, usa-se a rotina `void s_transstart(void)`. Este sensor usa um protocolo i2c um pouco diferente do habitual. Deste modo, temos:

```
s_transstart()
        SCL(1)
        SDA(0)
        SCL(0)
        SCL(1)
        SDA(1)
        SCL(0)
```

Prosseguindo, em caso de erro é necessário realizar o *reset* da comunicação com o sensor. Usa-se a rotina `void s_connectionreset()`.

```
s_connectionreset()
        SDA(1)
        SCL(0)
        Para(i=0 até i<9)
                SCL(1)
                SCL(0)
        s_transstart()    // dar novamente inicio à comunicação
```

Se a comunicação ocorrer sem erros, realiza-se o cálculo da temperatura e da humidade relativa usando a rotina `void calc_sth11(float *p_humidity ,float *p_temperature`).

Nesta rotina, realiza-se a conversão dos valores lidos do sensor para as unidades respectivas.

```
calc_sth11()
        C1=-4.0;
        C2= 0.0405;
        C3=-0.0000028;
        T1=0.01;
        T2=0.00008;
        temperatura= p_temperature *0.01-40.0; resultado em °C
        rh_lin=C3*p_humidity²+C2*p_humidity+C1;
        humidade=( temperatura-25)*(T1+T2*rh)+rh_lin;
```

Por fim, realiza-se o cálculo do *dewpoint* com a rotina `float calc_dewpoint(float h,float t)`

Esta função recebe como parâmetros a temperatura e a humidade e devolve o *dewpoint* Referência [36].

```
calc_dewpoint()
        k = (log10(h)-2)/0.4343 + (17.62*t)/(243.12+t);
        dew_point = 243.12*k/(17.62-k);
```

### 5.2.2 *Acelerómetro*

**Objectivo:**

Um acelerómetro permite detectar variações bruscas de velocidade. Pode-se usar um acelerómetro para medir aceleração assim como medir as mais variadas coisas, tais como, Aceleração, *Tilt*, ângulo de *Tilt*, inclinação, rotação, vibração, colisão,

gravidade. Todos estes dados podem ser extremamente úteis em certas aplicações Robóticas.

Os acelerómetros são já usados hoje em dia nas mais diversas máquinas e equipamentos especializados, tais como:

- Piloto automático de aviões "*Model airplane auto pilot*"
- Sistemas de Alarme de Automóveis
- Detecção de Acidentes, desenvolvimento de *airbags*
- Monitorização de Movimento
- Ferramentas niveladoras

Antigamente os acelerómetros tinham um comprimento grande, eram pesados, e acima de tudo bastante caros o que tornava impossível a sua utilização em projectos de Robôs e electrónica.

O Acelerómetro utilizado é o "*Memsic* 2125 Dual *Axis Accelerometer*" da "*Parallax*".

**Princípio de Funcionamento:**

Normalmente, as pessoas sentem a aceleração em três eixos, frente/trás, esquerda/direita e cima/baixo. Basta pensar na última vez que fomos passageiros num automóvel numa estrada sinuosa. Aceleração Frente/Trás, verifica-se quando se aumenta a velocidade e se abranda. Aceleração Esquerda/Direita, verifica-se quando curvamos para a esquerda ou para a direita. Aceleração Cima/Baixo, é o modo como sentimos a Força da Gravidade (por exemplo, numa lomba na estrada). Quando estamos no chão, as pessoas tendem a sentir a gravidade como sendo o seu próprio peso. Em queda livre, sente-se como sendo a falta de peso. Em termos físicos, Gravidade é a forma da aceleração. Quando um objecto está parado no chão, à sua gravidade chama-se aceleração estática. Quando o objecto se encontra em movimento, a gravidade transforma-se em aceleração dinâmica. Em vez dos três eixos que são sentidos pelas pessoas, o acelerómetro em questão, o "MX2125" sente a aceleração em dois eixos. A aceleração sentida, depende do modo como está posicionado. Colocando-o na horizontal, pode extrair-se Frente/Trás e Esquerda/Direita. Se o colocado na vertical, pode ler-se por exemplo Cima/Baixo e Frente/Trás. Normalmente usar um acelerómetro de dois eixos é suficiente para a maior parte das aplicações, contudo, pode sempre usar-se um segundo acelerómetro para se obter o terceiro eixo. Referência [15].

**Esquema Eléctrico:**

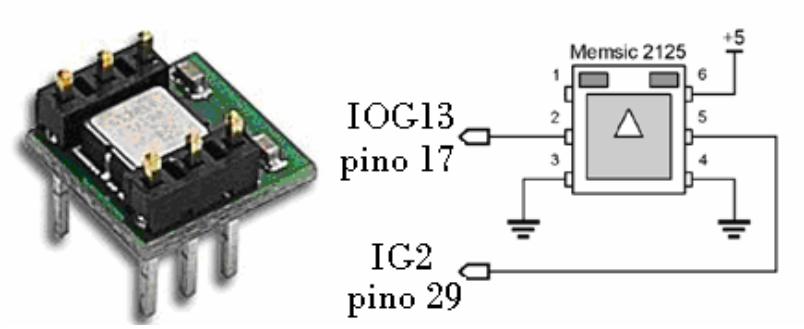


**Figura 30 Acelerómetro MX2125**

**Características técnicas:**

O acelerómetro em questão, é constituído por uma camera de gás, com um dispositivo que aumenta a temperatura no centro, formando uma bolha de ar quente e quatro sensores de temperatura nas extremidades. Partindo do princípio que o ar quente sobe, ao passo que o ar mais frio desce, quando o acelerómetro estiver colocado num plano, o ar quente fica no topo da camera e todos os sensores de temperatura medem o mesmo valor. Se inclinarmos o acelerómetro, o ar quente vai tender a deslocar-se para um dos lados. Deste modo, podemos sentir a aceleração.

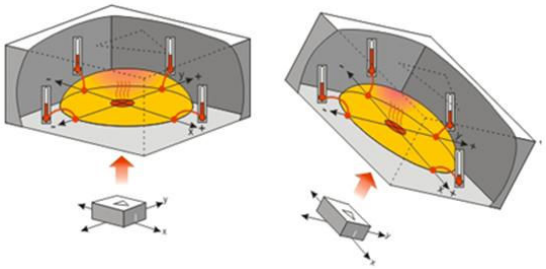

**Figura 31 Camera de gás Acelerómetro**

Tanto a aceleração estática (gravidade) como a aceleração dinâmica (viagem de automóvel) são detectadas pelos sensores de temperatura. O acelerómetro converte ainda os sinais lidos pelos sensores de temperatura em dois sinais com forma de onda quadrada, cujo *duty-cycle* varia com a aceleração sentida, é de 50% a 0g e varia proporcionalmente com a aceleração. Referência [14].

Como principais características técnicas, temos que:

- Capaz de medir de 0 a ± 2g em ambos os eixos, tendo uma resolução de 1mg;
- Capaz de funcionar correctamente entre temperaturas de 0º a 70º C;

- Como saída, fornece duas ondas quadradas onde o seu *duty-cycle* varia com a variação da aceleração em X e em Y;
- Saída analógica de temperatura;
- Consumos baixos de corrente, menos de 4mA a 5V;

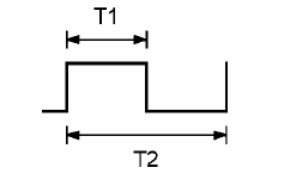


**Figura 32 Duty-Cycle**

Para calcular a aceleração, usa-se:

**Equação 12**

$$A(g) = ((T\_1/T\_2 - 0.5))/(12.5\%)$$

em que, a duração de T2 é de 10 milissegundos a 25ºC.

Para calcular o ângulo de inclinação para qualquer um dos eixos, usa-se a

**Equação 13**

$$inclinação = \sin^{-1}(A(g))$$

**Como é controlado:**

Este dispositivo é controlado através do controlador *FoxBoard*, que realiza a monitorização do estado dos pinos IOG13 e IG2 e para ambos faz o cálculo do *duty-cycle*, do valor de *A(g)* e do valor da inclinação do respectivo eixo.

**Software realizado:**

Foram realizadas duas rotinas, `void thread_AcelX(void)` e `void thread_AcelY(void)`, cujo algoritmo está apresentado na Figura 34, para realizar a monitorização constante de ambas as saídas do acelerómetro.

Inicialmente, a rotina criada testa uma *flag* global para verificar se chegou ao fim. Como o objectivo é calcular o *Duty-cycle* de uma onda quadrada, começou-se por procurar o início da onda, ou seja a primeira transição (0 → 1 ou 1 → 0). Quando essa transição ocorre, temos o primeiro instante (T0) de tempo. De seguida, procuram-se mais duas transições, de modo a ter um período completo da onda. Obtêm-se mais dois instantes de tempo (T1 e T2).

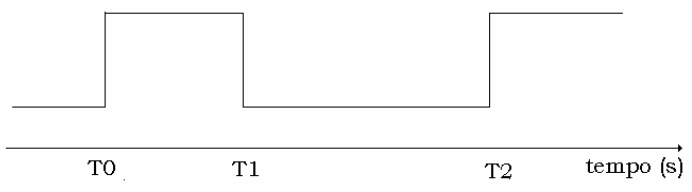


**Figura 33 Instantes de Tempo Duty-cycle**

Deste modo, fazendo *t1* = T1-T0, temos o tempo alto e fazendo *t2* = T2-T1, temos o tempo baixo. Para determinar qual o tempo alto e o tempo baixo, verifica-se qual o último valor lido. Por exemplo se o último valor lido for "1", *t2* é o tempo em baixo e *t1* o tempo em alto.

Calcula-se de seguida o *Duty-Cycle*, o *A(g)* e a *inclinação*:

```
DutyX_=((float)T1X/((float)T2X+(float)T1X));
gX=(((1-DutyX)-0.5)/0.125);
Tilt = asin((double)gX);
```

O valor de *Tilt* é obtido em radianos, executa-se de seguida a conversão para graus.

```
Tilt=(((180)× Tilt)/3.14);
```

Os valores obtidos para a inclinação, *Tilt* e para o *Duty-cycle* são variáveis globais.

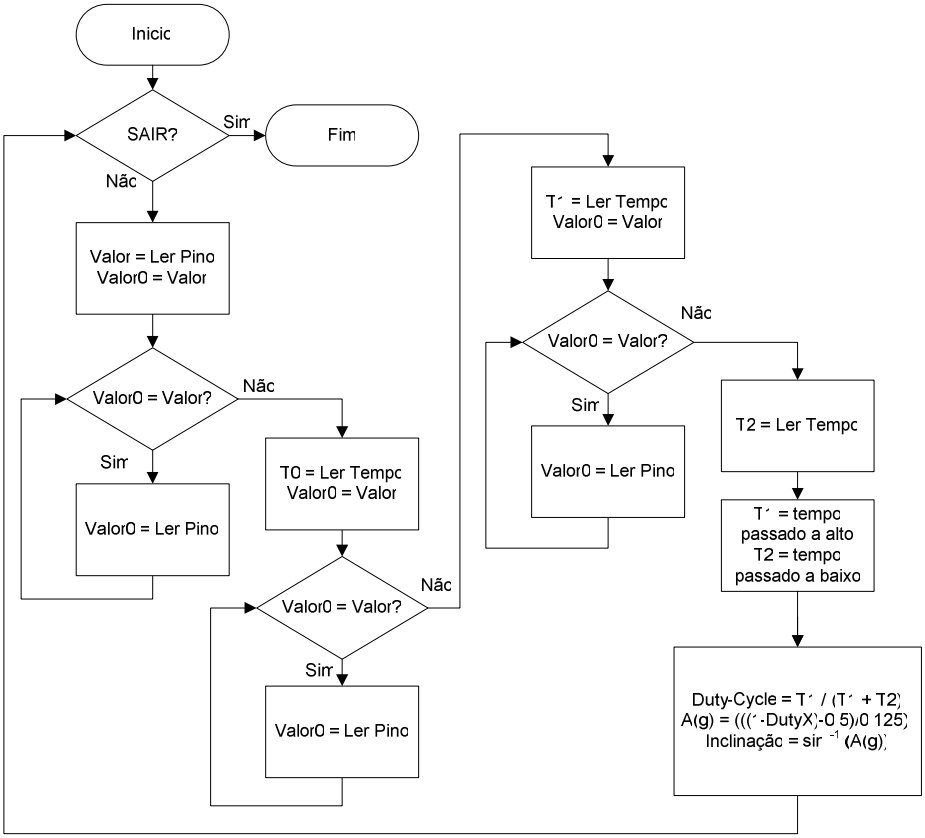


**Figura 34 Algoritmo Acelerómetro**

### 5.2.3 Encoders

**Objectivo:**

Permitir o cálculo da velocidade e ter noção da distância percorrida.

**Princípio de Funcionamento:**

O *encoder* é um dispositivo que mede um deslocamento (linear ou rotacional).

Encontra-se de algum modo em contacto com o eixo do motor, e quando o submetemos a rotação, o *encoder* origina um conjunto de sinais eléctricos por cada volta do seu eixo.

Os *encoders* possuem um conjunto de características bastante relevantes, que importa salientar, tais como a resolução, precisão, classe de precisão, entre outros.

A resolução, é o menor incremento que o *encoder* pode fornecer, ou seja, o número de pulsos emitidos por cada rotação. Quanto maior o numero de pulsos, maior a sua resolução.

A classe de precisão é a faixa de erro utilizada para classificar o *encoder*.

Temos ainda, que existem dois tipos de *encoders*, os incrementais e os absolutos. O *encoder* do tipo incremental somente gera sinais quando o seu eixo está em movimento. O *encoder* do tipo absoluto, gera um conjunto de bits e mesmo com o seu eixo parado, ou seja, uma máquina que possui *encoders* do tipo absoluto, não precisa de movimentar os seus eixos para obter informação dos *encoders*.

**Esquema Eléctrico:**

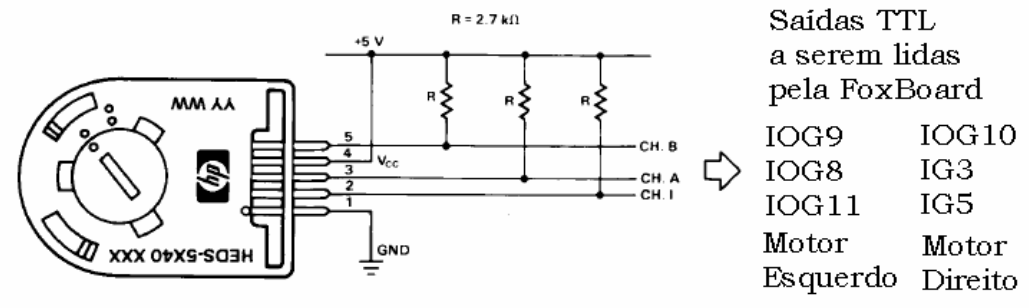


**Figura 35 Esquema Eléctrico Encoder**

Apresentam-se na Figura 35, as ligações efectuadas e os pinos da *FoxBoard* associados aos respectivos motores.

**Características técnicas:**

Os *encoders* HEDS-5540 são ópticos do tipo incremental de três canais com uma resolução de 512 pulsos por volta.

As formas de onda de saída do *encoder* são apresentadas na Figura 36. O canal A e o canal B apresentam uma onda quadrada desfasada, de forma a distinguir qual o sentido de rotação do *encoder*. Apresenta ainda um terceiro canal, canal I que dá um pulso por cada volta completa realizada.

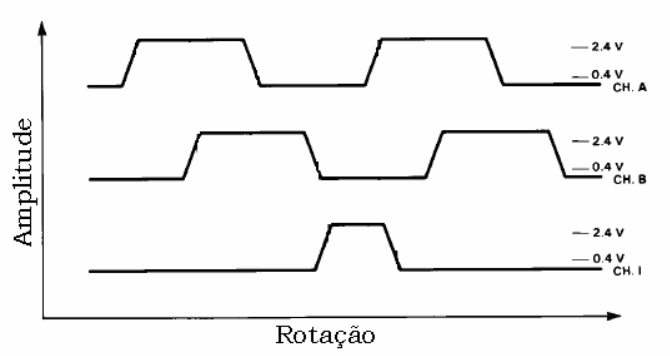


**Figura 36 Formas de onda de saída**

Apresentam-se na

Tabela 5, os valores das principais características do encoder. Referência [16].

**Tabela 5 Valores característicos do HEDS - 5540**

| Parâmetro | HEDS-55XX/56XX |
|---|---|
| Temperatura | -40°C a 100°C |
| $V_{CC}$ | -0.5 V a 7 V |
| VO | -0.5 V a $V_{CC}$ |
| $I_{OUT}$ | -1.0 mA a 5 mA |
| Velocidade | 30,000 RPM |
| Aceleração | 250,000 rad/sec$^2$ |

**Como é controlado:**

Os sinais emitidos pelos *encoders* são controlados pela FOXBOARD.

Realiza-se a leitura de um dos canais de cada *encoder*, para calcular o valor de velocidade aproximado do robô. Para o motor esquerdo, usa-se o pino IOG8 e para o motor direito usa-se o pino IG3 da FOXBOARD.

**Software realizado:**

Foram realizadas duas rotinas, `void thread_encoderE(void)` e `void thread_encoderD(void)`, semelhantes para os dois *encoders*, para realizar o cálculo da velocidade dos motores. O cálculo da velocidade é baseado na duração de um pulso.

A rotina testa uma *flag* global para verificar se chegou ao fim. De seguida, procura o início da onda quadrada e guarda os instantes em que ocorrem as transições.

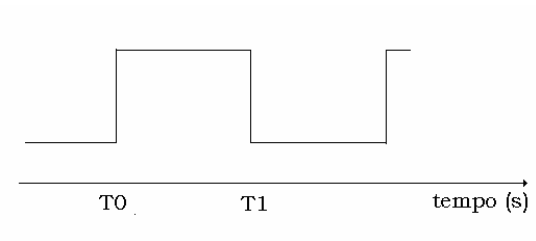


**Figura 37 Instante de tempo encoder**

Tendo os valores dos instantes em que ocorrem as transições (T0 e T1), calcula-se a duração do pulso (T1 – T0).

```
tempo_voltaE = curtimefE - curtimeE;
```

Tendo a duração de um pulso, sabendo que o *encoder* tem uma resolução de 512 pulsos por volta, calcula-se o tempo que demora a dar uma volta.

```
tempo_1voltaE = (512) * tempo_voltaE;
```

Por fim, sabendo que a velocidade é o quociente da distância sobre o tempo e que o perímetro da roda do robô é de $2 \times \prod \times R$, sendo o raio da roda = 20cm, temos:

```
velocidade_E = ((2*M_PI*0.2) / ((float)tempo_1voltaE);
```

A Figura 38 mostra um algoritmo, onde se pode observar o modo de funcionamento da função.

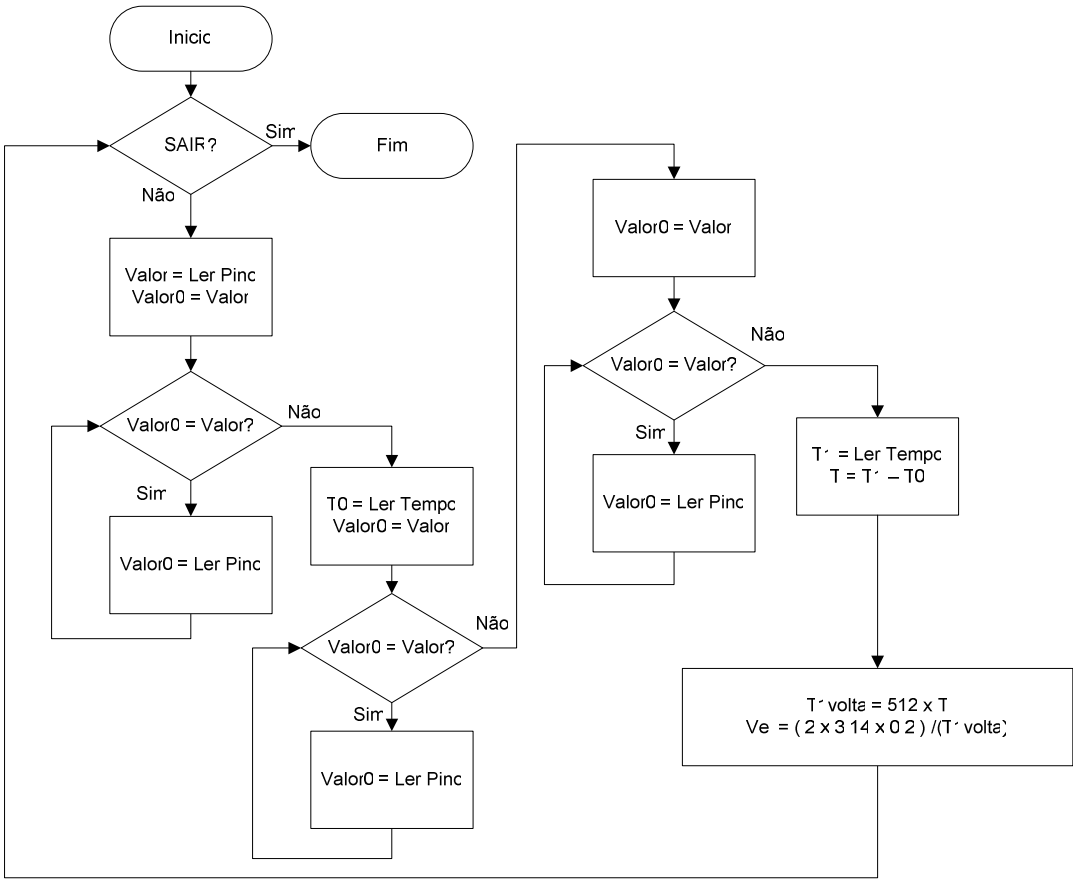


**Figura 38 Algoritmo velocidade *encoder***

Todos os ciclos, têm um contador a fazer como *timeout*, para o caso em que o robô está parado, actualizar a respectiva velocidade.

### 5.2.4 GPS

**Objectivo:**

Realizar o posicionamento absoluto do robô e de um alvo (localização desejada).

**Princípio de Funcionamento:**

É baseado no atraso do sinal, ou seja, o tempo que um sinal demora a percorrer uma distância. Sendo conhecida a velocidade de propagação do sinal e tempo, permite estimar a distância a que o receptor GPS se encontra do satélite.

Permite estimar a longitude e latitude, elevação, condição atmosférica e a hora actual em qualquer ponto na superfície da Terra.

**Esquema Eléctrico:**

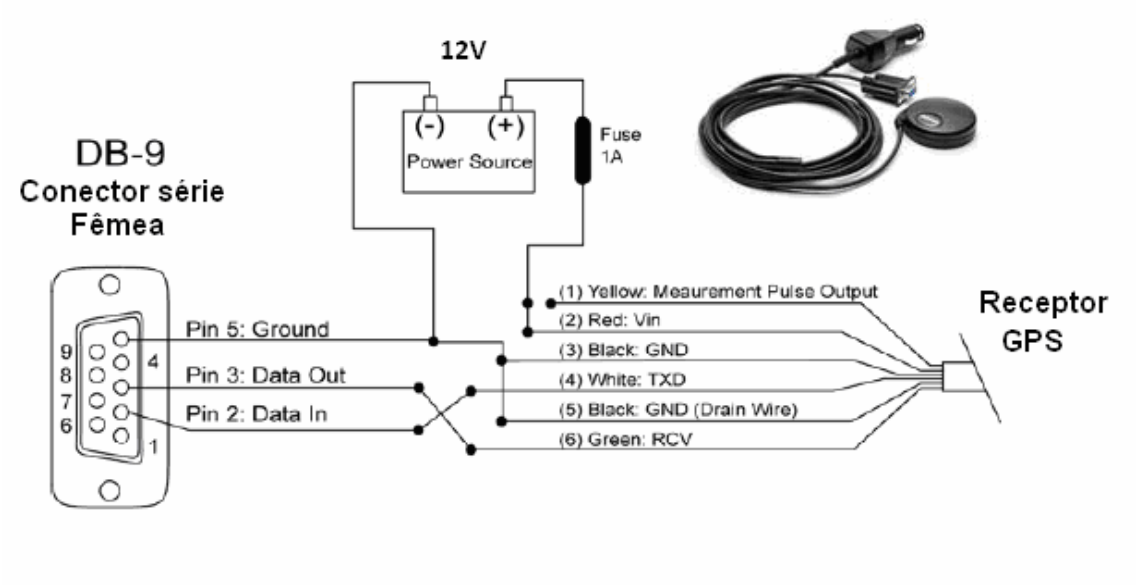


**Figura 39 GPS GARMIN 18PC**

O receptor GPS utilizado foi um modelo da *Garmin* 18PC, que usualmente é usado nos automóveis.

**Características técnicas:**

O GPS utilizado é um 18PC da *Garmin*, possui interface série, RS232 com um *baud rate* de 4800. Possui um conector para ser alimentado no isqueiro do automóvel.

Quando a 12V tem um consumo nominal de 80mA. A tensão de alimentação pode variar de 8 a 30V.

Suporta temperaturas de -30°C a +80°C em funcionamento e: -40°C a +90°C quando armazenado.

O tempo de actualização dos dados é de cerca de 1 segundo.

Usando o protocolo standard de posicionamento obtêm-se um erro menor de 15m.

Usando o sistema WAS, obtêm-se um erro menor de 3 m. Referência [37]

Possui a versão NMEA 0183 2.0 ou a NMEA 0183 2.30.

As tramas NMEA enviadas são: $PRMC, $GPGGA, $GPGSA, $GPGSV, $PGRME, $GPGLL, $GPVTG, $PGRMV, $PGRMF, $PGRMB e a $PGRMT.

São usadas as tramas $GPRMC, $PGRME e $GPGGA enviadas pelo receptor GPS. Estas são lidas através de umas das portas USB da FOXBOARD. Foi utilizado um

conversor RS232, uma vez que o GPS possui interface série. Note-se que poderíamos usar uma porta série disponível na FOXBOARD, só que, para o fazermos, é necessário desactivar umas das portas USB, assim como construir algum hardware para fazer a ligação.

Apresentam-se de seguida exemplos das tramas registadas e qual o significado dos seus campos. Referência [37]

```
$GPRMC,153717,A,4127.0527,N,00817.3940,W,000.0,293.7,151107,004.1,W*79
```

**Tabela 6 Trama NMEA GPRMC**

| $GPRMC | Identificação da trama – "*Recommended minimum specific GPS/TRANSIT data*" |
|---|---|
| 153717 | Tempo UTC 15:37:17 |
| A | A, posição válida; V, posição inválida |
| 4127.0527,N | Latitude, 41 graus, 27 minutos e 5,27 segundos Norte |
| 00817.3940,W | Longitude, 8 graus, 17 minutos e 39,40 segundos Este |
| 000.0 | Velocidade em nós[8] |
| 293.7 | Direcção viajada nos últimos segundos (gaus) |
| 151107 | Data UTC 15 de Novembro de 2007 |
| 004.1,W | Variação magnética, 4,1 graus Este |
| *79 | checksum |

Note-se que a latitude e a longitude vêm expressas em graus, minutos e segundos. Para se realizar Seguido o exemplo, temos:

Latitude = 41ºgraus 27' minutos 10,29 segundos

Longitude = 8ºgraus 17'minutos 51,8 segundos

Para converter o valor referido para graus, faz-se:

Latitude = 41 + 27/60 + 10,29/3600 = 41,45285833graus

Longitude = 8 + 17/60 + 51,8/3600 = 8,314388889graus

Posteriormente, estes valores vão ser usados para realizar a localização e controlo de trajectórias do robô.

```
$PGRME,10.0,M,12.3,M,16.0,M*18
```

[8] Nó é uma unidade de medida de velocidade equivalente a uma milha náutica por hora, ou seja 1852 m/h. É utilizada generalizadamente na navegação marítima e aérea

**Tabela 7 Trama NMEA PGRME**

| | |
|---|---|
| $PGRME | Identificação da trama – "Estimated position error" |
| 10.0,M | Erro estimado da posição horizontal em metros (HPE) |
| 12.3,M | Erro estimado da posição vertical em metros (VPE) |
| 16.0,M | Erro estimado da altura em metros ???? |
| *18 | Checksum |

$GPGGA,153750,4127.0531,N,00817.3931,W,1,04,2.6,218.4,M,52.0,M,,*5C

**Tabela 8 Trama NMEA GPGGA**

| | |
|---|---|
| $GPGGA | Identificação da trama – "Global Positioning System Fix Data" |
| 153750 | Tempo UTC 15:37:50 |
| 4127.0531,N | Latitude, 41 graus, 27 minutos e 5,31 segundos Norte |
| 00817.3931,W | Longitude, 8 graus, 17 minutos e 39,31 segundos Este |
| 1 | 0 informação inválida, 1 informação GPS válida, 2 informação DGPS válida. |
| 04 | Número de satélites presentes no horizonte |
| 2.6 | precisão relative da posição horizontal, "Horizontal Dilution of Precision (HDOP)" |
| 218.4,M | Altitude acima do nível do mar (metros) |
| 52.0,M | Peso do "geoid9" acima do elipsoide WGS84 |
| *5C | Checksum |

[9]Geoid - É a superfície equipotencial que, em média, coincide com o valor médio do nível médio das águas do mar.

### 5.2.5 Bússola

**Objectivo:**

Produzir um número para representar a direcção do robô.

**Princípio de Funcionamento:**

Existem materiais que se atraem ou repelem por possuírem uma força magnética, denominam-se de ímans. Estes possuem dois pólos, o Norte e o Sul que ficam em extremidades opostas. Quando se aproximam os pólos diferentes eles tendem a aproximar-se. Quando se aproximam os pólos iguais eles tendem a repelir-se.

A terra consiste num íman gigante e todos os imans à superfície da Terra são atraídos pelo pólo Norte e Sul do íman terrestre. É este o princípio de funcionamento de uma Bússola.

**Esquema Eléctrico:**

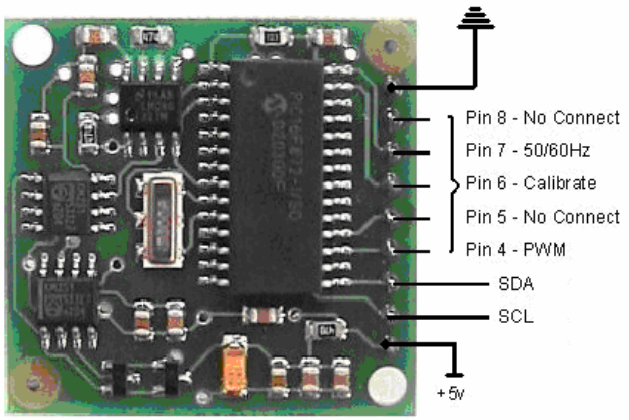


**Figura 40 Bússola**

**Características técnicas:**

Este dispositivo, foi especialmente desenhado para ser usado em robôs, para auxiliar a navegação.

O objectivo é produzir um número para representar a direcção do robô. Possui um sensor de campo Magnético (*Philips* KMZ51), que é capaz de detectar o campo magnético terrestre.

A interacção é realizada, através de um barramento i2c nos pinos 2 e 3, ou através de um sinal PWM no pino 4. Usando o barramento i2c, interage de imediato, seguindo o protocolo. Usando o modo PWM, 1ms corresponde a 0º e 36,99ms corresponde a 359,9º. A título de curiosidade, o PWM é gerado usando um TIMER de 16 bits que oferece uma resolução de 0,1º (10us). Referência [38].

**Como é controlado:**

Este dispositivo é controlado, usando o barramento *i2c* controlado pela FOXBOARD e possui o endereço (0XC0). De início, manda-se um *start* bit usando como endereço do dispositivo (0XC0), colocando o bit de leitura/escrita a zero, para poder aceder aos registos deste. De seguida, repete-se novamente o *start* e coloca-se o endereço no modo de leitura (0XC1). Por fim, pode-se realizar a leitura dos registos.

Os vários registos disponíveis para interacção, bem como as suas funções são apresentados na Tabela 9. Referência [38].

**Tabela 9 Registos usados pela Bússola**

| Registo | Função |
|---|---|
| 0 | Fornece qual a versão do programa. |
| 1 | Leitura da Bússola, 0-255 para uma volta completa. |
| 2,3 | Leitura da Bússola, 0-3599 para uma volta completa. |
| 4,5 | Teste Interno - Sensor1 |
| 6,7 | Teste Interno - Sensor2 |
| 8,9 | Teste Interno – Calibração valor 1 |
| 10,11 | Teste Interno - Calibração valor 2 |
| 12 | Não Utilizado – Lido como Zero |
| 13 | Não Utilizado – Lido como Zero |
| 14 | Não Utilizado |
| 15 | Registo de Calibração |

**Software realizado:**

Foi escrita uma função, `"void LerBussola(void)"`, cujo algoritmo é apresentado de seguida.

**Ler Bússola:**

```
Start
Endereço do dispositivo (0XC0)
Qual o registo que se pretende ler (0X01)
Start
Endereço do dispositivo no modo de leitura (0XC1)
Ler valor
Stop
```

### 5.2.6 Ultra-Sons

**Objectivo:**

Medir distâncias e detectar a presença de obstáculos.

**Princípio de Funcionamento:**

A medição é baseada em sinais acústicos – o sonar emite um sinal sonoro e faz a leitura do sinal reflectido, calculando o tempo entre a emissão e a recepção. Deste modo, quando detectado um obstáculo, conhecendo o tempo e a velocidade de propagação das ondas sonoras, é possível determinar a distância.

Este tipo de Sensores usa a emissão e a reflexão de ondas ultra sónicas que possuem frequências acima dos 20KHz e normalmente não podem ser ouvidas por Humanos. Existem na Natureza alguns animais que utilizam ultra-sons, tais como os morcegos, para detectar objectos e os golfinhos, para se comunicarem.

O ultra-som interage com os corpos sólidos e parte da energia da onda transmitida é reflectida de volta, como podemos verificar na figura seguinte.

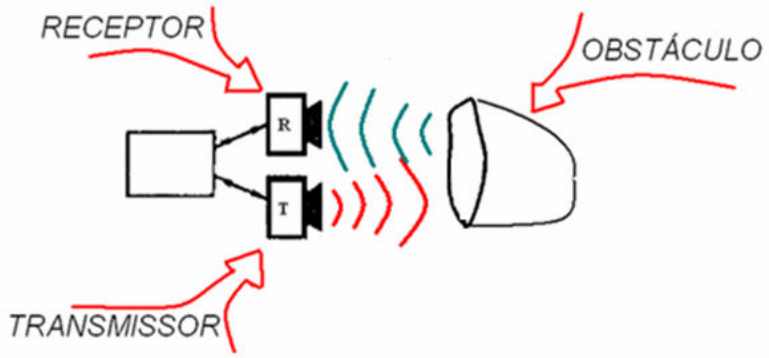


**Figura 41 Reflexão do ultra-som**

**Esquema Eléctrico:**

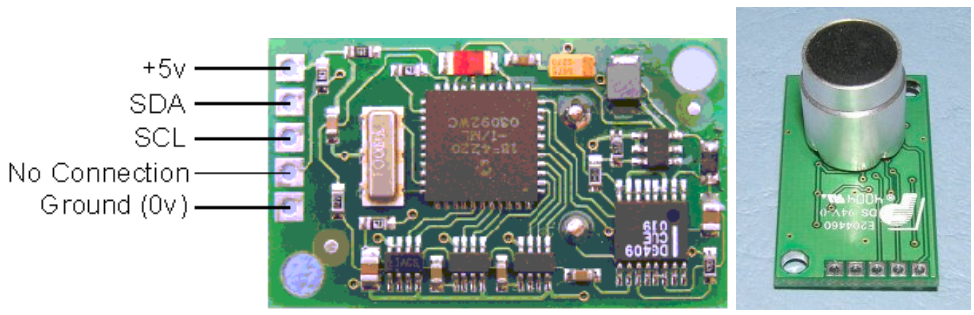


**Figura 42 Ultra-Som SRF235**

**Características técnicas:**

O Sensor ultra-som utilizado, foi o SRF235 e possui interface i2c, uma banda de alcance de apenas 15º, apresentada na Figura 43.

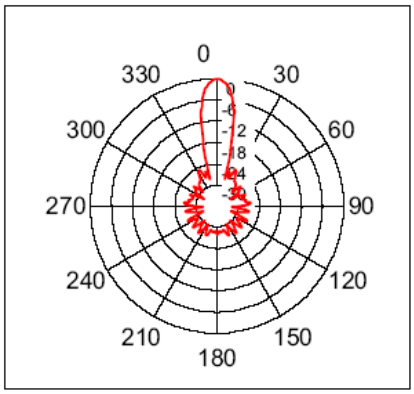


**Figura 43 Beam Pattern**

Este ultra-som emite frequências na ordem dos 235KHz e por isso o seu alcance é cerca de 1 metro não conseguindo as mesmas distâncias que os de 40KHz .

Este ultra-som é ideal para realizar um mapeamento detalhado do ambiente que rodeia o robô, conseguindo detectar com sucesso objectos.

Para conseguir detectar objectos a grandes distâncias, recomenda-se a utilização de outros modelos, tais como SRF10 ou SRF08, que são ultra-sons de 40KHz e possuem uma banda de alcance maior.

Usar uma combinação de ambos em torno de um robô para realizar o mapeamento do ambiente é o ideal. Referência [18]

**Como é controlado:**

O ultra-som utilizado é controlado através da FOXBOARD. Este está alojado no barramento *i2c*, funciona sempre como *slave* e tem o endereço (0XE0) por defeito.

Possui um conjunto de 4 registos

**Tabela 10 Registos usados pelo SRF 235**

| Registo | Função |
|---|---|
| 0 | *Registo de Comando* |
| 1 | Não usado |
| 2 | Byte mais significativo |
| 3 | Byte menos significativo |

Apenas se podem escrever comandos no registo 0. É utilizado para dar início a uma sessão de leitura. Cada sessão demora cerca de 10ms e não responde ao barramento

i2c enquanto este não terminar. Se for feita a leitura, obtêm-se a versão do software do *SRF235*.

Os registos 2 e 3 são utilizados para obter os resultados da sessão.

Apresentam-se na Tabela 11 os comandos que podem ser usados para interagir com o ultra-som. Referência [18]

**Tabela 11 Comandos SRF235**

| Comando | | Acção |
|---|---|---|
| Decimal | Hex | |
| 80 | 0x50 | Resultado – Polegadas |
| 81 | 0x51 | Resultado – Centímetros |
| 82 | 0x52 | Resultado - micro-segundos |
| 160 | 0xA0 | Mudar endereço I2C |
| 165 | 0xA5 | Mudar endereço I2C |
| 170 | 0xAA | Mudar endereço I2C |

**Software realizado:**

Foi criada uma função, `void Range_Sonar(unsigned char enderecoW, unsigned char enderecoR, unsigned char registoW, unsigned char registoR, unsigned char valor)`, cujo algoritmo é apresentado de seguida.

**Range Sonar:**

```
Start
Endereço do dispositivo (0XE0)
Qual o registo que se pretende escrever (0X00)
Escrever o valor (0X51) no registo seleccionado
Esperar 10ms
Start
Endereço do dispositivo (0XE0)
Qual o registo que se pretende ler (0X03)
Start
Endereço do dispositivo no modo de leitura (0XE1)
Ler valor
Stop
```

## 5.3. Actuadores

### 5.3.1 Placas de Controlo dos Motores

**Objectivo:**

O principal objectivo destas placas é realizar o controlo dos motores.

**Princípio de Funcionamento:**

A *MD03* é uma placa para realizar o controlo de motores de média potência. É constituída por uma ponte H controlada por PWM (*Pulse Width Modulation*) à frequência de 15KHz.

As pontes H são circuitos electrónicos que controlam motores de corrente contínua. O esquema básico é apresentado na Figura 44.

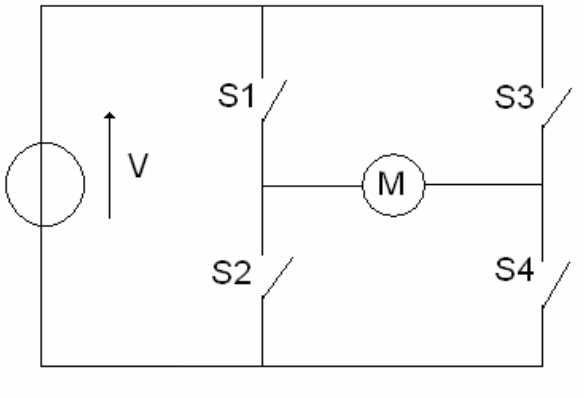


**Figura 44 Ponte H**

Basicamente, pode referir-se que são constituídas por quatro semicondutores de potência, *mosfets* do tipo n (para a *MD03*), cuja função é variar a tensão média aplicada aos terminais do motor.

Por exemplo, se os *mosfets* S1 e S4 estiverem fechados e os S3 e S2 abertos, vai ser aplicada aos terminais do motor uma tensão média positiva. Contrariamente se S1 e S4 estiverem abertos e S2 e S3 fechados, vai ser aplicada aos terminais do motor uma tensão média negativa, alterando o sentido de rotação do motor.

De notar que os *mosfets* não podem estar todos fechados ao mesmo tempo, porque originaria um curto-circuito aos terminais do motor.

A modulação por PWM está baseada numa onda quadrada cujo *duty-cycle* varia, como apresentado na Figura 45.

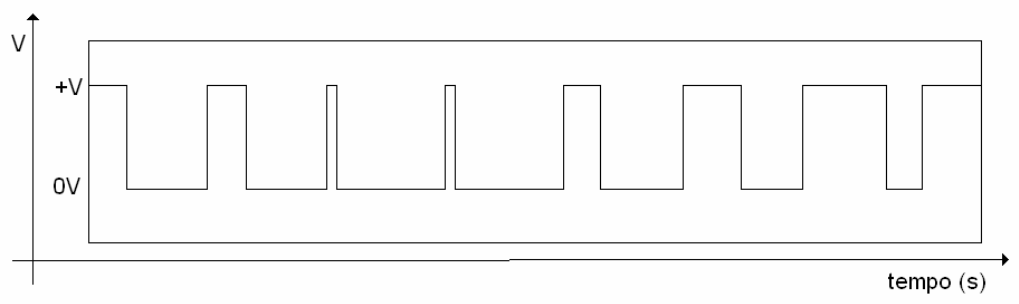


**Figura 45 Variação do Duty-Cycle**

A onda representada é aplicada a dois semicondutores, por exemplo S1 e S4 enquanto os outros dois estão abertos. Note-se que, alterando o *duty-cycle*, altera-se o valor médio da tensão +V aplicada ao motor. Deste modo, aumenta-se ou diminui a velocidade de rotação do motor.

**Esquema Eléctrico:**

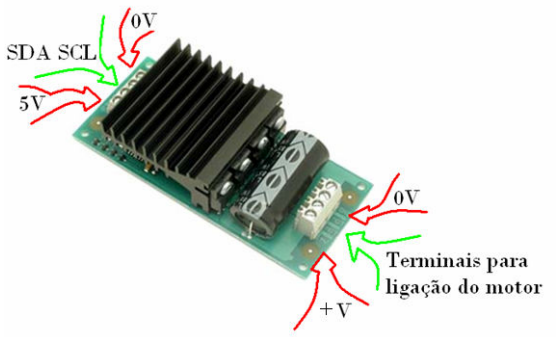


**Figura 46 MD03**

**Características técnicas:**

A *MD03*, possui uma parte lógica, na qual está ligado o barramento *i2c* e uma parte de potência onde está ligado o motor e a alimentação deste.

Necessita ainda de ser alimentada a 5V com um consumo máximo de corrente de 50mA para a parte lógica.

Consegue realizar o controlo de um número máximo de 8 motores por barramento *i2c*, cuja tensão varie de 5V a 50V.

Possui um LED vermelho que se liga sempre que o valor da corrente ultrapasse os 20A.

O controlo pode ser realizado de vários modos, tais como, recorrendo a um barramento *i2c*, através de valores analógicos de tensão, usando módulos RC ou um simples PWM.

Note-se ainda que é necessária protecção com um fusível de 20/30A, assim como eliminar ruídos com a adição de um condensador de 10nF, que podem ser prejudiciais ao correcto funcionamento dos restantes circuitos do sistema. Referência[17].

**Como é controlado:**

A *MD03* é controlada recorrendo ao barramento *i2c* da *FoxBoard*.

Na Tabela 12, apresentam-se os registos da *MD03*.

**Tabela 12 Registos *MD03***

| Registo | Nome | Leitura/Escrita | Descrição |
|---|---|---|---|
| 0 | Comando | L/E | 01 Sentido positivo - 02 Sentido negativo (00 paragem instantânea) |
| 1 | Estado | Leitura | Estado da Aceleração, Temperatura e Corrente |
| 2 | Velocidade | L/E | Velocidade Motor 0-255 (0x00 - 0xFF) |
| 3 | Aceleração | L/E | Aceleração do Motor 0-255 (0x00 - 0xFF) |
| 4 | Temperatura | Leitura | Temperatura |
| 5 | Corrente do Motor | Leitura | Corrente do Motor |
| 6 | Não usado | Leitura | Não usado |
| 7 | Versão do Software | Leitura | - |

No registo 0, diz-se qual o sentido que o motor deve rodar. 01 para positivo e 02 para negativo. Serve ainda para realizar paragem de emergência escrevendo 00.

No registo 1, pode verificar-se qual o estado dos parâmetros, aceleração, temperatura e corrente. O bit 0, tem o valor lógico "1", enquanto o motor estiver a acelerar. O bit 1 , tem o valor lógico "1", quando a corrente excede o valor máximo de 20A. O bit 2 tem o valor lógico "1", quando a temperatura chega a um valor exagerado. Nesta situação, a potência fornecida ao motor é limitada e o LED vermelho é activado. Se continuar muito tempo nesta situação, apesar de a placa limitar a potência fornecida ao motor, pode sofrer um sobreaquecimento e danificar-se.

O registo 2 serve para alterar o valor da velocidade. Aplica-se um valor de 0 a 243, consoante a velocidade desejada.

O registo 3 serve para alterar a taxa com que o motor acelera ou desacelera desde a velocidade actual para a velocidade desejada. Aplicam-se os valores desde 0 a

255. Quanto maior for o valor aplicado, mais tempo vai demorar a atingir a velocidade desejada.

O registo 4, serve para limitar internamente a corrente aplicada ao motor se a temperatura da placa for muito elevada. Pode realizar-se a leitura desse valor apesar da informação não ser muito relevante, na medida em que, o valor obtido não vem em graus. Na realidade cada unidade equivale a 1.42ºC.

O registo 5, serve para limitar internamente a corrente da placa. Ao realizar a leitura deste valor, obtêm-se um valor proporcional à corrente do motor. Tem-se que o valor 186 representa o limite de 20A.

Os registos 6 e 7 não são utilizados. Referência[17].

**Software realizado:**

Foi realizada uma função para escrever nos registos das placas MD03 `void md03_i2c(unsigned char endereco, unsigned char registo, unsigned char valor).`

```
Md03_i2c:
    Start
    Endereço do dispositivo
    Qual o registo que se pretende escrever
    Escrever o valor no registo seleccionado
    Stop
```

De salientar que existem duas *MD03*, uma para cada motor, usando o endereço (*0XB2)* para o motor direito e (*0XB0)* para o motor esquerdo.

Na função apresentada, faz-se o *start*, precedido do endereço do dispositivo, selecciona-se o registo a escrever, e realiza-se a escrita do valor no respectivo registo. Por fim, liberta-se o barramento *i2c* fazendo *stop*.

Esta função deu origem a uma outra, `void anda(int veloD, int veloE )` que recebe como variáveis os valores de velocidade para os motores. O objectivo é actuar sobre as placas que controlam os motores, escrevendo valores de direcção e velocidade. A velocidade controla-se, alterando o valor escrito no registo *(0X02)* e a direcção no registo *(0X00)*.

```
anda:
 para o motor direito
      escrever qual o sentido
      escrever a velocidade
 para o motor esquerdo
      escrever qual o esquerda
      escrever a esquerda
```

Realizou-se ainda a rotina, `LerMD03(unsigned char endereco, unsigned char endereco1, unsigned char registo)` para realizar a leitura de valores presentes nos registos das MD03 .

**Ler Md03:**

```
Start
Endereço do dispositivo
Qual o registo que se pretende ler
Start
Endereço do dispositivo no modo de leitura
Ler o valor do registo seleccionado
Stop
```

Dá-se o sinal de início, selecciona-se qual o endereço do dispositivo, coloca-se no barramento qual o registo a ler, repete-se o *start*, selecciona-se o dispositivo novamente, só que agora no modo de leitura, realiza-se a leitura do valor presente no registo previamente seleccionado e por fim realiza-se o *stop*.

Esta função deu origem a uma outra, `void LerTem_Current()`, cujo objectivo é monitorizar os valores da corrente e temperatura das placas em cada instante.

**Ler Temperatura e Corrente:**

```
Ler Corrente motor direito
Ler temperatura motor direito
Ler Corrente motor esquerdo
Ler temperatura motor esquerdo
```

O valor da corrente é lido no registo (0X05) e o valor da temperatura é lido no registo (0X04).

### 5.3.2 Motores

Um motor eléctrico converte a energia eléctrica em mecânica.

Nos robôs móveis usam-se usualmente motores de corrente contínua (CC), porque são fáceis de controlar e podemos usar como fonte de energia uma vulgar bateria.

Os motores escolhidos para o projecto, são motores normalmente utilizados em cadeiras de rodas, potência e binários consideráveis, encaixando no perfil do robô.

**Objectivo:**

O principal objectivo do uso de motores é transformar energia eléctrica em energia mecânica, originando assim um deslocamento permitir que o robô se desloque.

**Princípio de Funcionamento:**

O motor de corrente contínua é constituído pelos estátor e pelo rotor.

O estátor pode ser composto por um imã permanente ou por uma estrutura ferromagnética com pólos salientes onde são enroladas duas bobines que formam um campo magnético, como no caso da Figura 47.

O rotor é um electroíman constituído por enrolamentos em torno de um núcleo de ferro. O rotor é alimentado através de um sistema de escovas.

Como pólos diferentes se atraem e pólos iguais se repelem, consegue-se assim gerar o movimento do rotor. Quando se alimenta o rotor, cria-se um campo electromagnético que interage com o campo magnético permanente do estátor.

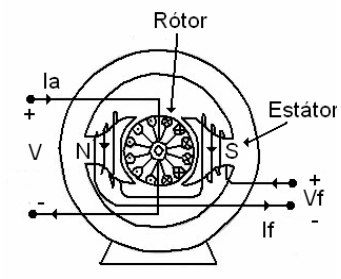


**Figura 47 Motor de corrente contínua de dois pólos**

Na Tabela 13 apresentam-se algumas das características do motor escolhido.

**Tabela 13 Características nominais do motor**

| Características: | |
|---|---|
| Potência nominal | 180 W |
| Tensão nominal | 24 V |
| Corrente nominal | 7.5 A |
| Binário | 14.0 Nm |
| Velocidade | 115 rpm |
| Eficiência à velocidade e binário nominais | 46 % |
| Corrente de pico | 40 A |
| Binário máximo | 55 Nm |
| Caixa de redução | 1/32 |
| Binário mínimo de travagem | 2 Nm |
| Corrente máxima gerada na travagem | 0.4 A |
| Peso | 6 kg |

**Como é controlado:**

A velocidade do motor é controlada, através do PWM aplicado ao rotor, gerado pelas placas de controlo dos motores, *MD03*.

## 5.4. Comunicação

Para realizar a comunicação entre os vários dispositivos, podemos usar os mais variados protocolos e sistemas. Apresentam-se de seguida alguns dos sistemas utilizados para os vários dispositivos comunicarem entre si, *i2c* e porta série, e para realizar a comunicação com o exterior através de um Joistick remoto, módulo de XBee.

### 5.4.1 Barramento i2c

No barramento *i2c*, estão colocados vários dispositivos, os quais é necessário controlar, escrevendo e lendo valores nos seus diversos registos, continuamente.

É então, criada na `main()`, a função `void thread_i2c(void)`.

Note-se que existem acções a realizar com prioridade sobre outras, como é o caso da leitura dos valores da Humidade comparando com a actualização dos valores da velocidade nas placas de controlo dos motores.

Esta função corre em ciclo infinito, para assim permitir uma actuação constante.

Apresenta-se de seguida o algoritmo realizado.

```
thread_i2c:
        LerBusola
        Se (Flag ultrasons = 1)
        Range_Sonar(0xE0,0xE1,0x00,0x03,0x51)
        anda(velocidadeD,velocidadeE)
        i++
        Se (i = 300)
        Calcula_Humidade
              LerTem_Current
              mostra_variaveis
              i = 0
```

Esta função vai percorrer todos os dispositivos ligados fisicamente no barramento *i2c* e realizar a sua monitorização e controlo.

Dá-se início por realizar a leitura da direcção actual do robô, usando a bússola. De seguida, se por pedido, verifica-se se estão presentes obstáculos e posteriormente, ordena-se às placas de controlo dos motores, *MD03* que coloquem os motores a andar à *velocidadeD*, para o motor direito, e à *velocidadeE* para o motor esquerdo. Estes valores de velocidade são calculados previamente em funções que correm em paralelo com esta.

Existem ainda dispositivos que não é necessário ler repetidamente. Estabeleceu-se então um valor para actualização destas variáveis. São estas a humidade, a temperatura e os valores de corrente dos motores.

### 5.4.2 Porta Série

A interface série, também conhecida como RS-232 é uma porta de comunicação utilizada para conectar os mais diversos dispositivos, tais como modems, algumas impressoras e outros equipamentos de hardware.

Neste protocolo de comunicação, os bits são transferidos em série, ou seja, um bit de dados de cada vez (comunicação assíncrona). Existe um pino para enviar bytes de dados e outro para receber.

**Figura 48 RS232 Porta série**

Para duas portas comunicarem uma com a outra, a velocidade de comunicação e o modo de codificação dos dados tem que ser igual em ambas. Temos que existem quatro parâmetros que é necessário configurar, a velocidade de transmissão da porta, o comprimento da palavra, verificação da paridade, e o *stop bit*.

A velocidade de transmissão denomina-se *baud rate* e representa o número de bits transmitidos por segundo.

O comprimento da palavra, especifica o número de bits que compõem cada unidade de dados.

Temos ainda a verificação da paridade, que serve para verificar se houve erro durante a transmissão de erros. Quando adicionada, adiciona 1 bit à unidade de dados. Pode ainda ser paridade par, impar ou sem paridade.

Em suma, o *stop bit*, que indica o fim da transmissão da palavra de dados.

Na presente aplicação, usa-se o RS232 como interface com módulo de XBee e com o receptor de GPS.

Todo o código realizado para a leitura ou escrita da porta série foi baseado na Referência [22].

### 5.4.3 XBee

A comunicação sem fios, existe actualmente das mais diversas maneiras. A mais recente e promissora, utiliza o protocolo ZigBee [10] IEEE 802.15.4 e encontra-se em desenvolvimento pela ZigBee Alliane, http://www.zigbee.org/en/index.asp.

**Objectivo:**

O principal objectivo da utilização deste transmissor de rádio frequência é poder realizar o controlo remoto do robô recorrendo a um *JOYSTICK* desenvolvido pela SAR, assim como realizar rádio localização.

**Princípio de Funcionamento:**

O seu princípio de funcionamento está baseado no protocolo ZigBee

Os vários módulos podem ser ligados entre si em várias topologias, tais como Malha ou Ponto-a-Ponto, Árvore e Estrela.

**Esquema Eléctrico:**

Apresenta-se na Figura 49, um módulo de *XBee*, assim como quais as ligações a efectuar para realizar a ligação à porta série *ttyS2* da *FoxBoard*.

[10] ZigBee, ou IEEE 802.15.4, designa uma tecnologia de redes sem fios ainda em fase de desenvolvimento que pretende realizar a interligação de pequenas unidades de comunicações de dados em áreas muito limitadas

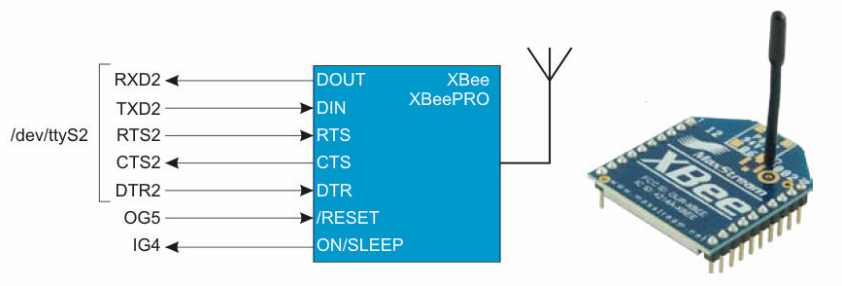


**Figura 49 Módulo XBee**

Apresentam-se de seguida algumas das suas principais características:

- Alcance em ambientes internos/zonas urbanas: 100m
- Alcance de RF em linha visível para ambientes externos: 1,6Km
- Taxa de transferência de dados dados de RF: 250.000 bps
- Tensão de alimentação: 2.8 a 3.4V
- Temperatura de operação: -40 a 85ºC
- Erro: Retransmite novamente (*Retries*) e reconhecimento (*acknowledgement*)
- Topologia de Rede: ponto-a-ponto, ponto-a-multiponto e malha
- Funciona na gama de frequência de 2.4000 - 2.4835 GHz

Referência [24].

Para realizar a conexão entre a FoxBoard e o módulo XBee foi usado o módulo apresentado na Figura 50, FoxZB.

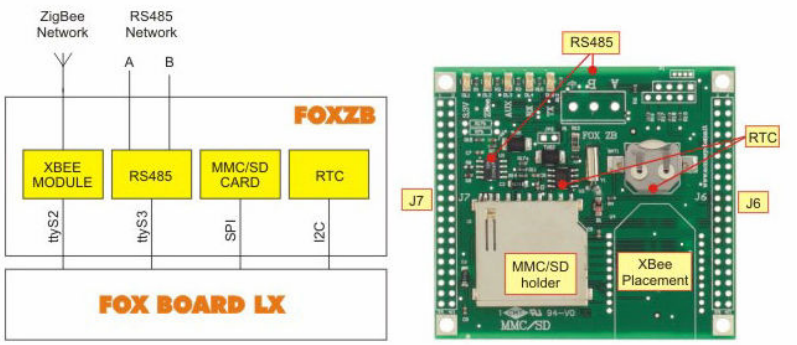


**Figura 50 FoxZB**

A utilização deste módulo veio permitir uma expansão do grupo de periféricos disponíveis na unidade de processamento contínuo *FoxBoard*.

Este módulo, desenhado para ser acoplado à *FoxBoard,* permite usar uma unidade de memória no formato de cartão MMC[11], possui o encaixe correcto para o módulo de XBee, possui um relógio em tempo real e possui ainda uma interface série RS485 Referência [23].

## 5.5. *JOYSTICK*

Foi realizada uma função `void thread_Joystick(void)`, que actualiza constantemente os valores desejados de velocidade para ambos os motores, provenientes do *JOYSTICK*.

Na `main( )` do sistema, com a rotina `void startjoystic()` são inicializados os parâmetros da porta série e activação do módulo de XBee para activar a sua comunicação com o JOYSTIC.

Nesta rotina, realiza-se o *reset* do módulo, colocando o pino 30 da FOX BOARD (IG5).

```
iomask=1<<5;
ioctl(i2c_fd,_IO(ETRAXGPIO_IOCTYPE,IO_SETBITS),iomask);
```

De seguida realiza-se a leitura do valor de *Baud Rate*, e inicializam-se os parâmetros do módulo de XBee.

Apresenta-se de seguida o algoritmo implementado.

```
Ler dados da porta série
vel = y*-2;
dir1 = x*-0.6;
Se(vel < 20 && vel > -20)
     veld = 0 ;
Senão
     veld = vel;
Se(dir < 10 && dir > -10)
     dird = 0;
Senão
     dird = dir;
limita_vel(dird,veld);
```

[11] MultiMediaCard - Um padrão de cartão de memória.

Os valores lidos na porta série são convertidos para valores que podem ser aceites pelo sistema.

Como o *Joystick* tem muita resolução, sempre que os valores recebidos sejam maiores ou menores que 20 valores, o robô segue em frente.

## 5.6. Alimentação

Apresenta-se na Tabela 14 um resumo de algumas características dos vários componentes do sistema.

**Tabela 14 Consumos de energia nominal dos vários componentes**

| Componente | Quant. | Energia | Humidade | Temperatura |
|---|---|---|---|---|
| sensirion STH11 (Sensor de Humidade) | 1 | 5V DC; 4mA; 20mW | - | -40ºC ~125ºC |
| Heds (Encoders Ópticos) | 2 | 5VDC | - | -40ºC~100ºC |
| MD03 (Placa de controlo dos motores) | 2 | 5V DC; 50mA e max de 50V 20A, 100W | - | - |
| Linksys (Router Wirelsess) | 1 | 3,3V DC; 2ª | 20%~80% | 0ºC~40ºC |
| PC (unidade processamento imagem) | 1 | 12V DC; 4.43ª | 10%~90% | 0ºC~60ºC ; |
| Fox Board (unidade de processamento contínuo) | 1 | 5V DC; 280mA; 1,4W | - | 0ºC ~70ºC |
| CNB-B2310PVF (câmara) | 1 | 12V DC; 0,5A; 6W | - | -10ºC ~ 50ºC |
| SRF 235 (Ultra-som) | 1 | 5V DC; 25mA | - | - |
| Garmin 18 PC (GPS) | 1 | 12V DC; 8mA | - | -30ºC ~ 80ºC |
| Acelerómetro | 1 | 5V DC; 4mA | - | 0º ~ 70ºC |
| Motor CC | 2 | 24V DC; 7,5A; 180W | - | - |
| Módulo Xbee | 1 | 2,8 a 3,4V DC | - | -40ºC ~ 85ºC |

### 5.6.1 Baterias

Um robô autónomo necessita de uma fonte de alimentação (potência), capaz de armazenar energia, para o robô realizar a tarefa para a qual foi construído. São usadas normalmente baterias. Uma bateria converte energia química em energia eléctrica.

Apresentam-se de seguida algumas das principais características das baterias.

**Densidade de Energia (Wh/kg)** – É a máxima quantidade de energia armazenada por quantidade de massa.

**Capacidade (Amp-hora ou mA-h)** - É a quantidade de energia armazenada na bateria. É o produto da densidade de energia multiplicado pala massa da bateria.

**Voltagem (V)** – Valor de tensão disponível aos terminais da bateria.

**Resistência Interna (Ω)** - Valor de resistência medido aos terminais de uma bateria.

**Vida útil** – Com o tempo, as baterias mesmo sem estarem ligadas a uma carga, vão perdendo energia. Com o aumento do número de cargas e descargas as baterias perdem capacidade de armazenamento de energia.

Uma bateria ideal tem que possuir densidade de energia elevada, conseguir manter uma tensão constante e ter um valor baixo de resistência interna.

Foram usadas baterias da marca DIAMEC, DMU 12-26 (12V 26AH/20Hr), com um ciclo de uso de 14,4V – 15V e corrente inicial menor que 7,8A.

### 5.6.2 Sistema de Alimentação

Apresenta-se na Figura 51 o sistema de alimentação desenvolvido para todo o sistema.

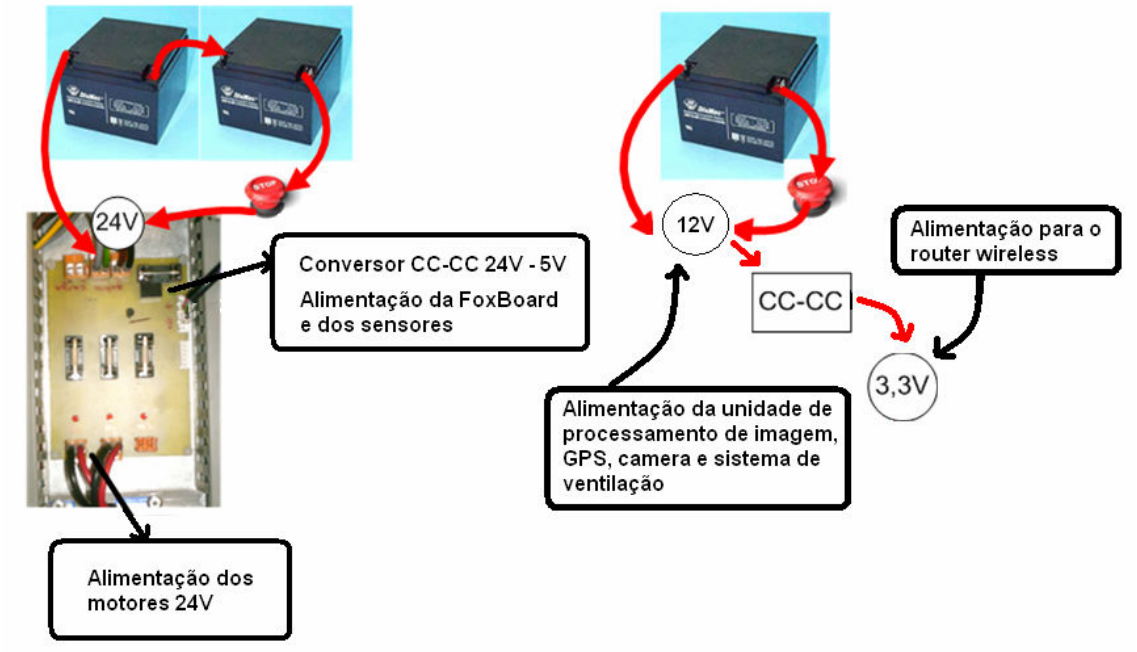


**Figura 51 Sistema de Alimentação do robô**

Note-se que, os dispositivos usados não possuem todos as mesmas características energéticas. Por exemplo, os motores têm que ser alimentados a 24V, as placas de controlo dos motores, a *FoxBoard* e a bússola a 5V, o receptor GPS e o computador a 12V e o *router wireless* a 3,3V.

Em colaboração com a empresa SAR, foi realizado o sistema de alimentação apresentado. É constituído por fusíveis adequados para a alimentação dos motores e para a alimentação dos vários dispositivos que são alimentados a 5V.

Note-se ainda que, possui sinais a serem lidos pela FoxBoard pelos pinos IOG22, IOG20, IOG18, IOG16 e IOG17 para indicar qual o estado dos fusíveis e se a placa possui energia ou não.

Para conseguir 3,3V para alimentar o *router wireless*, usa-se um conversor CC-CC apresentado na .Figura 52.

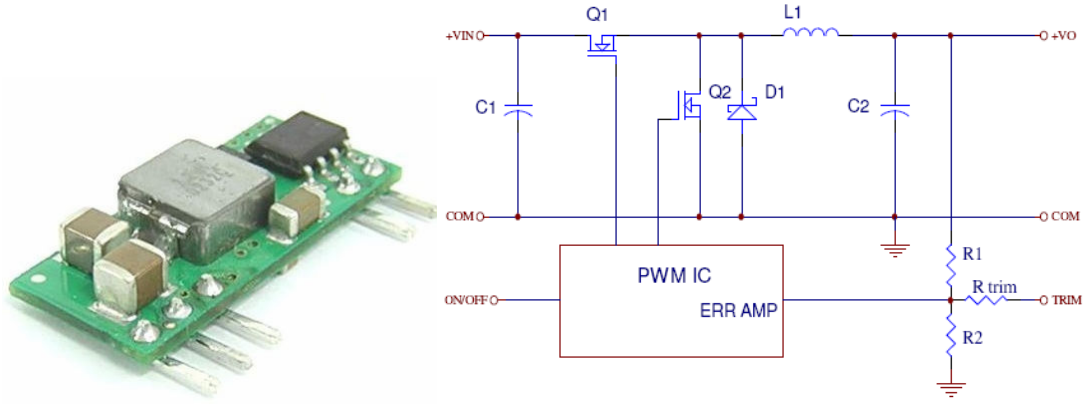


**Figura 52 Conversor CC-CC Diagrama Eléctrico**

Trata-se de um dispositivo cujo valor da tensão de entrada pode ser entre 8,3 e 14V e tensão de saída entre 0,75 a 5V com uma corrente máxima de 5A. Pode funcionar em ambientes cuja temperatura esteja entre -40ºC e 84ºC e humidade entre 20 e 95 %.

Possui protecção contra curto-circuito e contra valores excessivos de sobre corrente.

Trata-se de um dispositivo que oferece uma eficiência de 89%, o que leva a que o valor de energia libertada sob a forma de calor seja baixa, não necessitando de dispositivos para realizar a dissipação de calor. Referência [39].

Note-se que, é necessário ajustar a tensão de saída, adicionando uma resistência *Rtrim* ao circuito, como apresentado na Figura 53. Se esta resistência não for adicionada, a tensão de saída é por defeito, 0,75V.

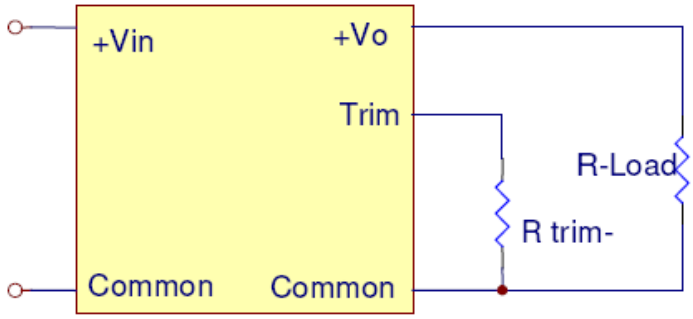


**Figura 53 Esquemático do conversor CC-CC com resistência Rtrim**

Para realizar o cálculo do valor de *Rtrim*, faz-se:

**Equação 14**

$$Rtrim = \frac{10500}{Vo - 0{,}75} - 1000$$

Tendo que, *Vo* é o valor da tensão de saída desejado.

Como se pretende um valor de 3,3V à saída, fez-se:

$$Rtrim = \frac{10500}{3{,}3 - 0{,}75} - 1000 = 3118\Omega$$

Para obter 5V, foi usado o dispositivo apresentado na Figura 54.

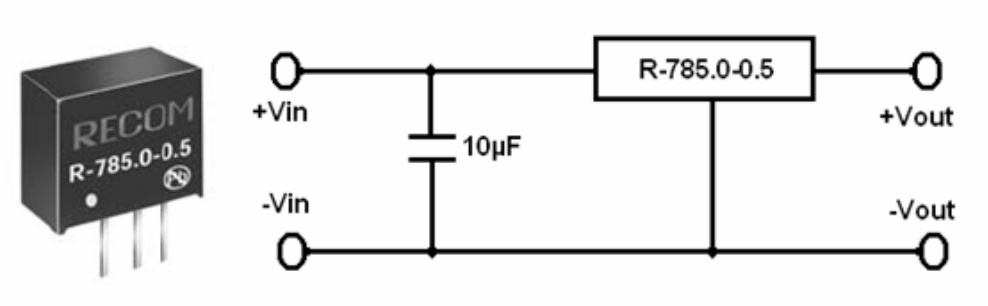


**Figura 54 RECOM – Conversor CC-CC**

Trata-se de um conversor CC-CC não isolado que possui uma eficiência acima dos 97%. O valor da tensão de entrada pode ser entre 6,5-34V, originando uma tensão de saída é de 5V com uma corrente máxima de 500mA. Possui um ripple baixo e protecção contra curto-circuito. Devido à sua elevada eficiência, a energia desperdiçada em forma de calor é muito pouca, não necessitando de dissipadores de calor. Funciona correctamente em ambientes cuja temperatura varie entre -40ºC a 85ºC. Referência [40].

# 6. Controlo

## 6.1. Controlo da velocidade dos motores

Para realizar o controlo da velocidade do robô, foi realizado controlo Proporcional Integral (P.I.), para cada motor.

Foram aplicadas as regras de *Ziegler-Nichols* para determinar valores aproximados das constantes *Kp* e *KI*.

Note-se que os resultados obtidos estiveram longe dos resultados finais, na medida em que, são inúmeros os parâmetros que influenciam o controlo da velocidade dos motores. Temos a irregularidade do terreno, a variação das condições de atrito devido ao estado da relva, a presença das bolas, buracos realizados pelas toupeiras, entre outros.Para ajustar os valores dos controladores usou-se o método da tentativa erro, dando valores e observando qual o comportamento do robô.

Os valores obtidos para os valores de KP e KI foram 1 e 0,05 respectivamente, para ambos os motores.

Apresenta-se o algoritmo de controlo de velocidade implementado.

```
Controlo PI
erroD = velD - velocidadeD
Se((erroD < 5) && (erroD > -5))
        erroD = 0
        erroD_anterior = 0
        velocidadeD0 = velD;
velocidade_auxD = velocidadeD0 + KPD*erroD + KID*erroD_anterior
erroD_anterior = erroD
Se(velocidade_auxD > 250.0)
        velocidadeD = 250
Se(velocidade_auxD < -250.0)
        velocidadeD = -250
Se(velocidade_auxD < 250.0 && velocidade_auxD > -250.0)
        velocidadeD = velocidade_auxD
 velocidadeD0 = velocidadeD
```

Para realizar o controlo da velocidade do robô foi aplicado o algoritmo acima descrito a cada um dos motores.

De início calcula-se o erro, e posteriormente dá-se uma tolerância. Usou-se o valor cinco que foi achado experimentalmente. Se a velocidade actual for maior ou menor que cinco, o erro tem um valor aceitável e anula-se.

De seguida, o valor de velocidade a incrementar ao valor actual somando o resultado de `KPD*erroD + KID*erroD_anterior`. Note-se que, se o erro for nulo o valor de velocidade actual vai ser igual ao valor de velocidade anterior.

De seguida, iguala-se o valor da variável que armazena o valor do erro anterior á variável que armazena o valor de erro actual.

De seguida, antes de aplicar directamente o resultado do controlo á placa de controlo dos motores, tem que se realizar uma limitação dos valores. Deste modo, a rotina que actua nas placas de controlo somente podem receber valores que variem desde -255 a 255.

## 6.2. Navegação

### 6.2.1 Localização

Para realizar o posicionamento do robô no espaço, é útil ter coordenadas cartesianas X e Y em vez de latitude e longitude.

Como o espaço em questão é de dimensão bastante pequena quando comparado com o raio da terra, podemos considerar um plano, em vez de considerar que a terra tem uma forma esférica.

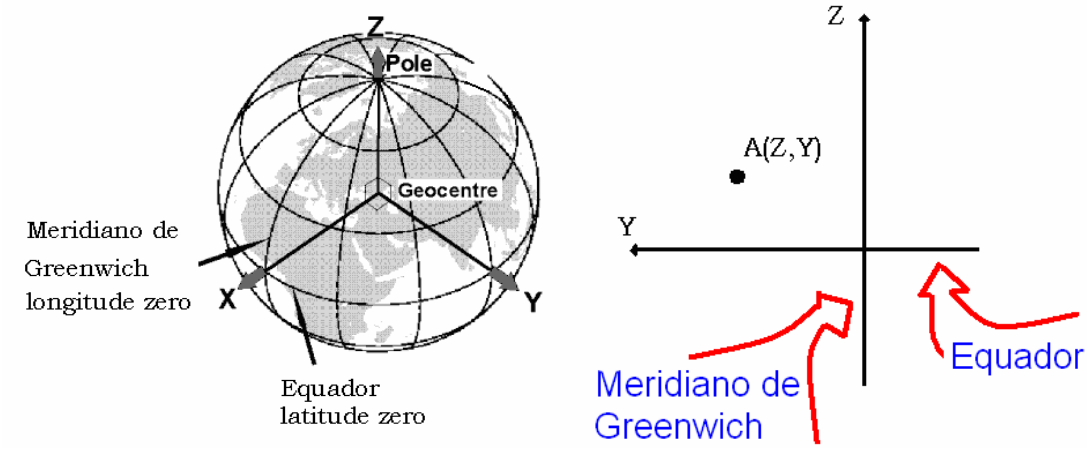


**Figura 55 Posicionamento**

Considerando somente o plano ZY, e por exemplo o ponto A, o local onde foi anotada a trama acima descrita, temos:

Latitude = 41 + 27/60 + 10,29/3600 = 41,45285833graus

Longitude = 8 + 17/60 + 51,8/3600 = 8,314388889graus

Usando a transformação de coordenadas polares para coordenadas cartesianas anteriormente descrita, temos:

Z = latitude = 6378.0 × sin(41,45285833) = 4222,258962Km

Y = longitude = 6378.0 × cos(41,45285833) × sin(8,314388889) = 691,2560407Km

Deste modo, como temos um ponto num plano, podemos analisar a sua posição com outro ponto.

### 6.2.2 Direcção

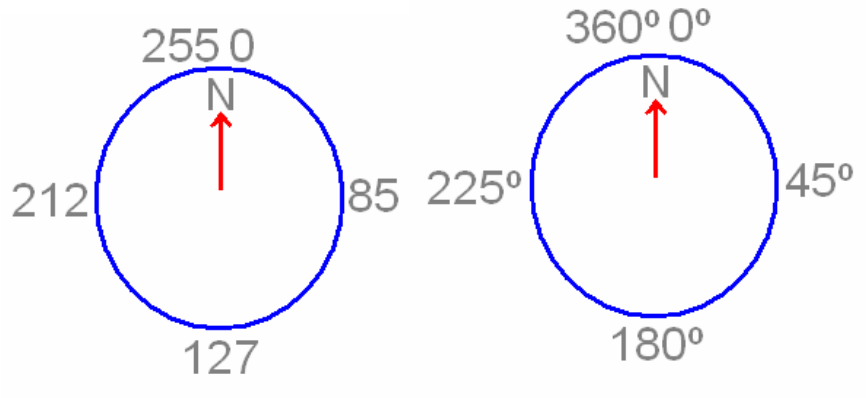


**Figura 56 Valores da Bússola**

Conhecendo a posição do robô e a posição do alvo, assinalado com X na Figura 57, realiza-se o cálculo do angulo θ, tendo como referência o Norte.

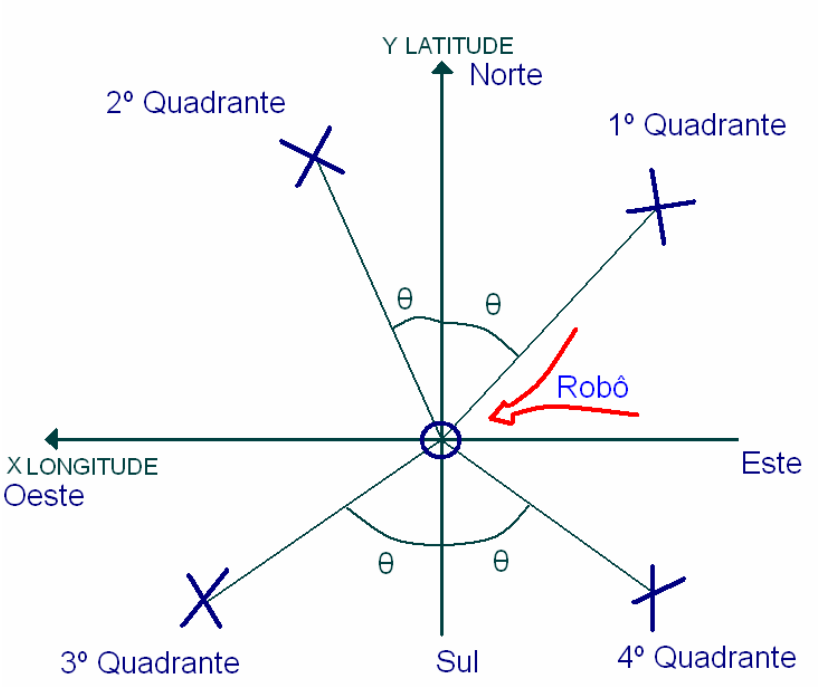


**Figura 57 Ângulo desejado em relação ao Norte**

O valor de Φ é calculado, fazendo:

**Equação 15**

$$\emptyset = \tan^{-1}\frac{dx}{dy}$$

Usando a bússola do robô e calculando o valor de bússola desejado, actua-se na variável que controla a direcção, de forma a posicionar o robô no sentido pretendido.

Para calcular o valor da bússola é necessário verificar o quadrante em que o alvo está em relação ao robô. É necessário ajustar o seu valor, numa gama de 0 a 360 º tendo como sentido positivo a orientação dos ponteiros do relógio, sendo o zero o ponto mais próximo do Norte e tendo em conta que o domínio de `atan()` varia de -pi/2 a pi()/2.

Como a gama da bússola do robô varia desde 0 a 255, no sentido dos ponteiros do relógio, é necessário realizar a conversão.

Cálculo do valor de direcção desejado, Algoritmo:

```
void thread_Actualiza_GPS(void)
```

**Actualiza_GPS(void)**

```
dx = lon_f - lon_km;
dy = lat_f - lat_km;
dt = ((sqrt(dx*dx + dy*dy))*1000.0);
Se dx > 0 && dy > 0                              //2° Quadrante
      bussola = 360 - atan(dx/dy);
Se dx > 0 && dy < 0                              //3° Quadrante
      bussola = 180.0 - atan(dx/dy);
Se dx < 0 && dy < 0                              //4° Quadrante
      x = dx * (-1);
      bussola = atan(dx/dy);
Se dx < 0 && dy > 0                              //1° Quadrante
      bussola = 180.0 - atan(dx/dy);
erroBGPS = bussola - Bussola;
Se (erroBGPS > -10 && erroBGPS < 10)
      dirG = 0;                              //Segue em frente
Se (erroBGPS > 10)
       dirG = dirG - 40;            //virar para a direita
Se (erroBGPS < -10)
       dirG = dirG + 40;            //virar para a esquerda
Se (dt> 10)
      Flag_comunica1 = 1;     //Desactivar Comunicação com o PC
      limita_vel(dirG,vel); //actualizar o valor da direccao
Senão
      Flag_comunica1 = 0;     //Activar Comunicação com o PC
```

Deste modo, realiza-se o cálculo do valor da Bússola desejado, ou seja, valor a seguir para atingir o alvo, em relação ao Norte.

De seguida, realiza-se a conversão de valores, para uma escala de 0 a 255 como fornecido pela Bússola, para realizar a comparação dos valores.

Para proceder à recolha de bolas no espaço em questão, dividiu-se o mesmo por zonas, como apresentado na Figura 58.

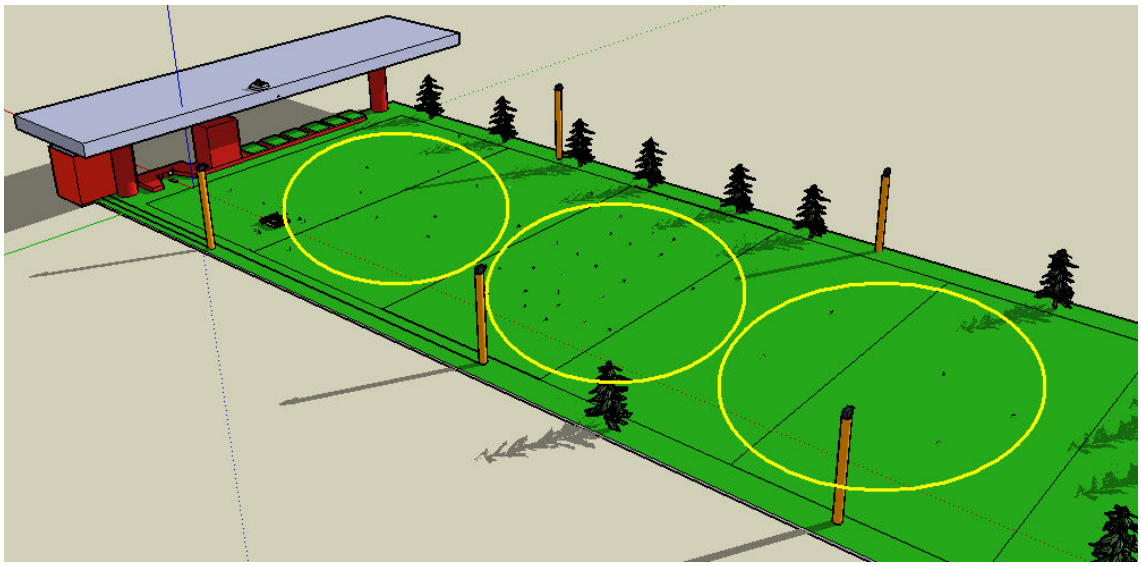

**Figura 58 Localização no espaço**

O objectivo é realizar uma prévia análise do terreno e realizar a recolha de pontos, coordenadas latitude e longitude no centro das circunferências assinaladas, determinar ainda qual a distância de segurança, (distância do centro do ponto em relação á rede ou a outro obstáculo), para cada uma das zonas e ordenar ao robô para se dirigir para a zona desejada definido trajectórias a percorrer.

Quando o robô está a uma distância *dt*, menor que distância de segurança do ponto desejado em relação à berma, activa-se o controlo por processamento de imagem e desactiva-se o controlo por GPS. Contrariamente, quando a distância *dt* é ultrapassada, activa-se o controlo por GPS e desactiva-se o controlo por processamento de imagem, por forma ao robô dirigir-se para o ponto desejado novamente, evitando assim a berma. Deste modo, é varrida a zona em questão.

Quando tiver passado um determinado tempo, ou por ordem do utilizador, o robô passa para uma próxima zona.

Quando tiver varrido todas as zonas, ou por ordem do utilizador, volta para o ponto de partida.

## 6.3. `Main()` do sistema

Esta é a rotina principal do sistema. Nela são inicializadas as funções que controlam todo o sistema.

Realiza-se a abertura do ficheiro "`/dev/gpiog`" para permitir o uso de pinos.

```
fdPin = open("/dev/gpiog", O_RDWR);
```

Realiza-se a abertura do ficheiro "/dev/ttyS2" para habilitar a porta série à qual está ligado o *JOYSTICK*.

```
fd = open("/dev/ttyS2", O_RDWR | O_NOCTTY | O_NDELAY);
```

Para activar a porta USB0 para ler o sensor GPS.

```
fdGPS = open("/dev/ttyUSB0", O_RDWR | O_NOCTTY | O_NDELAY);
```

Criação de uma *thread* para percorrer todo o barramento *i2c* continuamente actuando sobre os diversos dispositivos.

```
pthread_create(&thread_das_leituras, NULL, (void *)thread_i2c, NULL);
```

Criação de duas rotinas de monitorização constante dos *encoders* para realizar o cálculo do valor de velocidade.

```
pthread_create(&thread_do_encoder_E,NULL,(void*)thread_encoderE,NULL);
pthread_create(&thread_do_encoder_D,NULL,(void*)thread_encoderD,NULL);
```

Criação de uma rotina para ler a informação recebida do *Joystick*.

```
pthread_create(&thread_do_Joystick,NULL,(void *)thread_Joystick,NULL);
```

Criação de uma rotina para ler a informação recebida da unidade de processamento de imagem.

```
pthread_create(&thread_da_comunicacao,NULL,(void*)thread_comunica,NULL);
```

Criação de duas rotinas de monitorização constante dos dois sinais provenientes do acelerómetro.

```
pthread_create(&thread_da_ACELX, NULL, (void *)thread_AcelX,NULL);
pthread_create(&thread_da_ACELY, NULL, (void *)thread_AcelY,NULL);
```

Criação de uma rotina para realizar a monitorização constante do estado dos fusíveis.

```
pthread_create(&thread_dos_Fusiveis,NULL,(void*)thread_Actualiza_Fusiv
eis,NULL);
```

Criação de uma rotina para realizar a monitorização da posição global do robô e controlo da direcção a tomar do mesmo.

```
pthread_create(&thread_do_GPS,NULL,(void*)thread_Actualiza_GPS,NULL);
```

Por fim, possui ainda, um switch, que permite o controlo remoto do robô a partir do computador, assim como activar ou desactivar algumas tarefas ou acções a realizar pelo robô.

Para habilitar o controlo remoto, depois de executar a aplicação na FoxBoard, pressiona-se a tecla ‘J’ para desactiva o Joystick. Este inicia-se por defeito, aquando do arranque do sistema.

De seguida, usam-se as teclas ‘Q’ e ‘A’ para o robô andar para a frente e para trás respectivamente. Usam-se ainda as teclas ‘O’ e ‘P’ para virar para a direita e para a esquerda.

Usa-se ainda a tecla ‘C’ para habilitar o envio da informação dos sensores para a unidade de processamento de imagem e a tecla ‘Y’ para habilitar o controlo por esta unidade.

Por fim, possui ainda a tecla ‘G’ para activar o controlo recorrendo ao GPS.

# 7. Casos Práticos

Neste capítulo, apresentam-se alguns dos testes realizados para a análise de algumas das variáveis do sistema.

## 7.1. Pitch & Roll

Para testar os valores do acelerómetro, realizou-se uma viagem com o robô pelas zonas mais inclinadas do terreno.

Apresentam-se no Gráfico 1 os valores registados, para a variação da inclinação em graus.

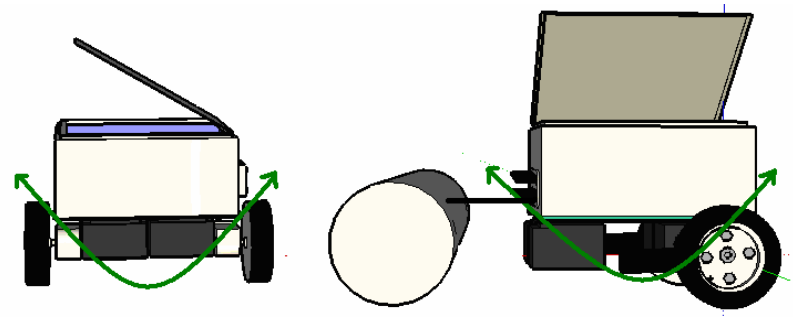

**Figura 59 Roll e Pitch**

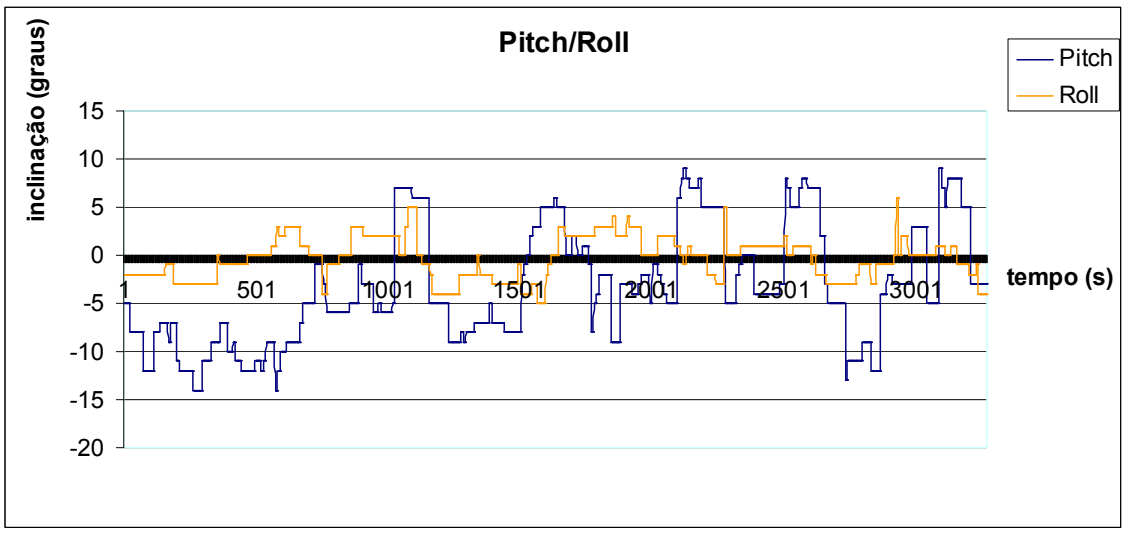


**Gráfico 1 Pitch e Roll**

Apresentam-se na Tabela 15 o valor médio, máximo e mínimo para o *Pitch* e o *Roll*.

**Tabela 15 Pitch e Roll**

| Pitch | | Roll | |
|---|---|---|---|
| Media | -3,344542953 | Media | -0,358300214 |
| Máximo | 9 | Máximo | 6 |
| Mínimo | -14 | Mínimo | -5 |

## 7.2. Corrente e temperatura dos motores

Para testar a leitura dos valores presentes nos registos da MD03, temperatura e corrente do motor, foram realizados vários testes e armazenando os dados em cada instante.

Apresentam-se no Gráfico 2 os valores de corrente dos dois motores recolhidos numa zona plana, sem qualquer tipo de controlo, em que é ordenado ao robô virar para a esquerda.

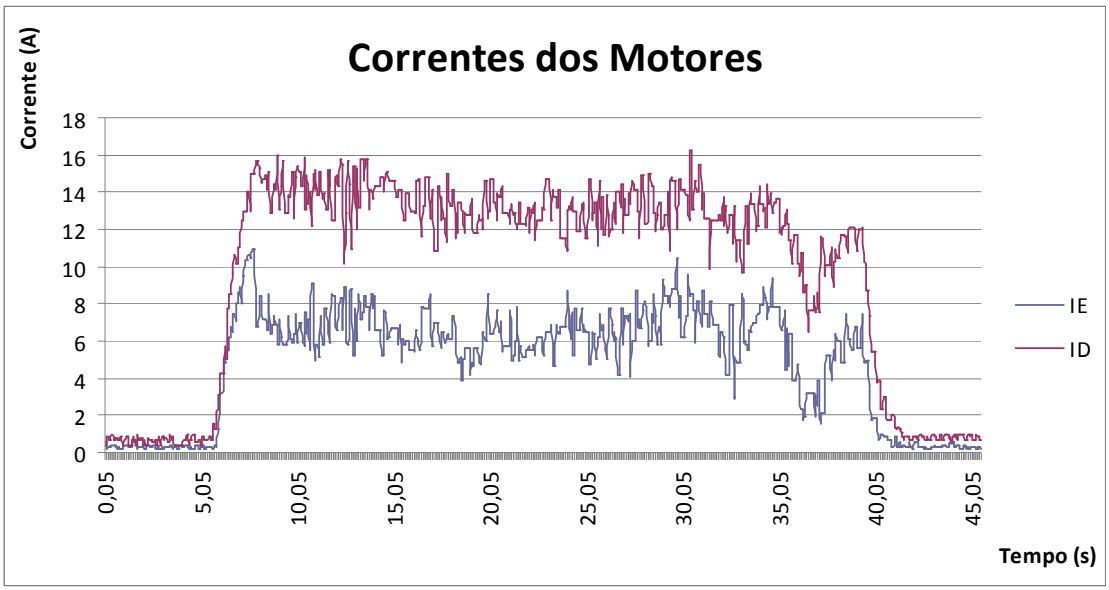


**Gráfico 2 Diferença de correntes nos motores**

Analisando o Gráfico 2 e a Tabela 16, verifica-se que os valores das correntes dos motores esquerdo e direito são diferentes, porque o robô está a virar para a esquerda, deste modo, o motor direito tem que possuir uma velocidade de rotação superior à do motor esquerdo, logo consumir um valor maior de corrente.

**Tabela 16 Valores das Correntes dos motores**

| Para o motor direito (A) | |
|---|---|
| Valor Médio | 12,5 |
| Máximo | 16,3 |
| Mínimo | 0,43 |
| **Para o motor esquerdo (A)** | |
| Valor Médio | 5,91 |
| Máximo | 10,9 |
| Mínimo | 0,22 |

Apresenta-se no Gráfico 3 e na Tabela 17, os valores de temperatura lidos no registo 4 de ambas as MD03. O valor varia de 0 a 255, sendo que cada unidade equivale a 1.42ºC. Note-se que quanto mais pequeno for o valor, maior é o valor da temperatura.

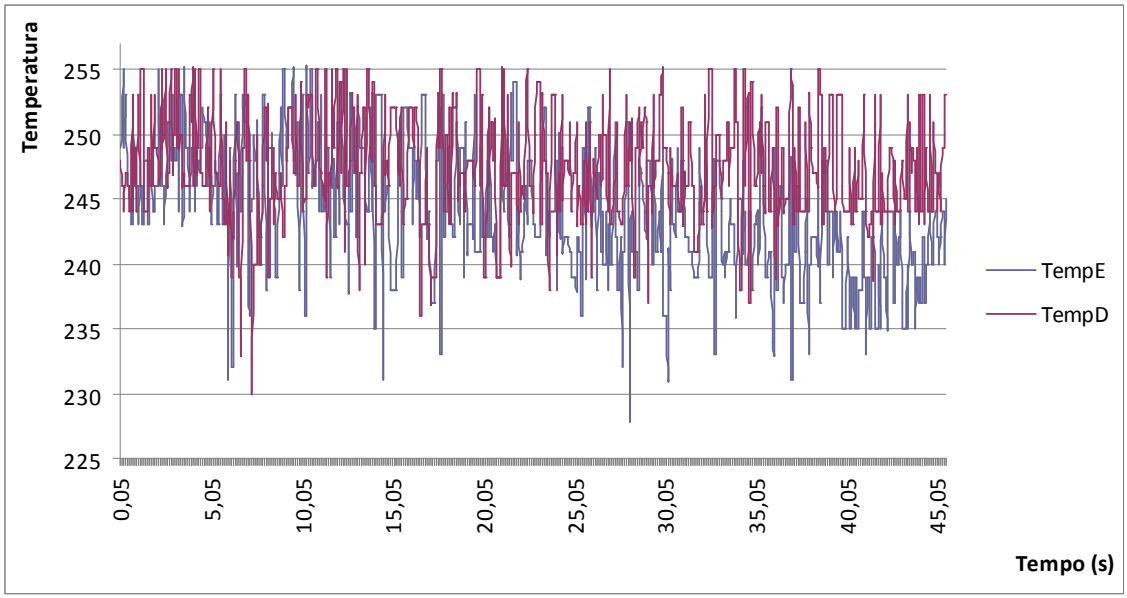


**Gráfico 3 Temperatura MD03**

**Tabela 17 Estatística da temperatura das placas de controlo dos motores**

| Para o motor direito | |
|---|---|
| Valor Médio | 247 |
| Máximo | 255 |
| Mínimo | 230 |
| **Para o motor esquerdo** | |
| Valor Médio | 244 |
| Máximo | 255 |
| Mínimo | 228 |

Realizou-se ainda um teste com várias paragens e arranques alterando a direcção e sentido do robô. Colocou-se o robô numa zona em que o terreno é inclinado. Apresenta-se no Gráfico 4, os valores de corrente consumidos pelos motores que foram registados.

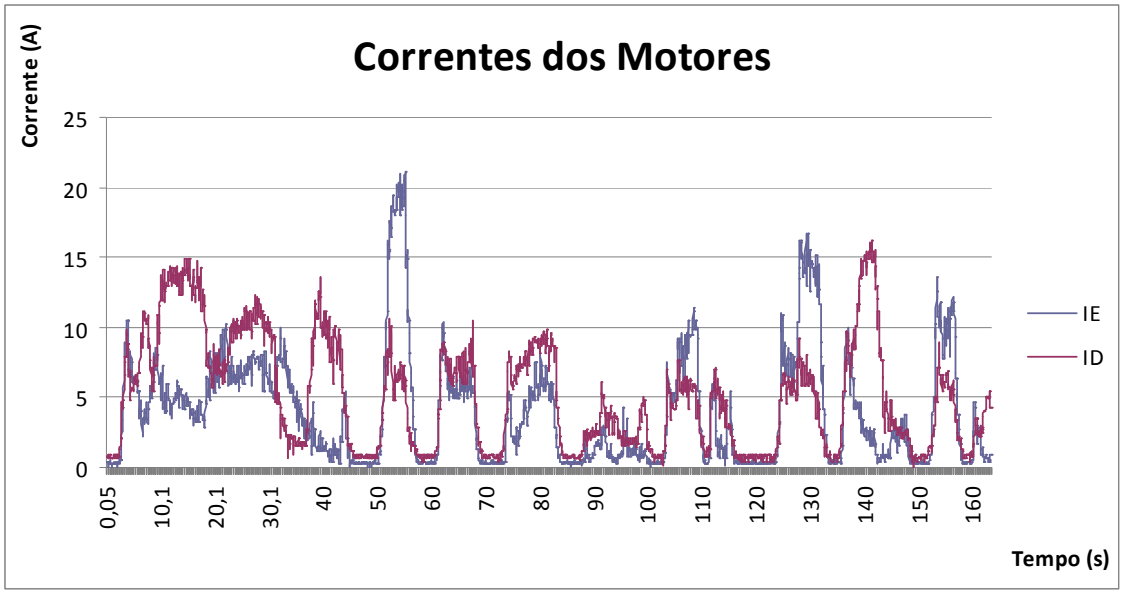


**Gráfico 4 Correntes dos motores**

Note-se que existem vários picos de corrente para ambos os motores, que representam os vários arranques realizados pelo robô.

Note-se ainda que, quando o valor de corrente de um dos motores é maior em relação ao outro, o robô está a curvar.

Por exemplo, quando o valor de corrente do motor da esquerda é maior que o da direita, o robô está a curvar para a esquerda e vice-versa.

## 7.3. Velocidade do Robô

Usando os *encoders*, obtém-se uma estimativa da velocidade do robô, para realizar o controlo.

Note-se que este valor é modificado através de uma interpolação, para ser comparado com o valor desejado da velocidade. Apresenta-se no Gráfico 5, um conjunto de valores obtidos para a velocidade.

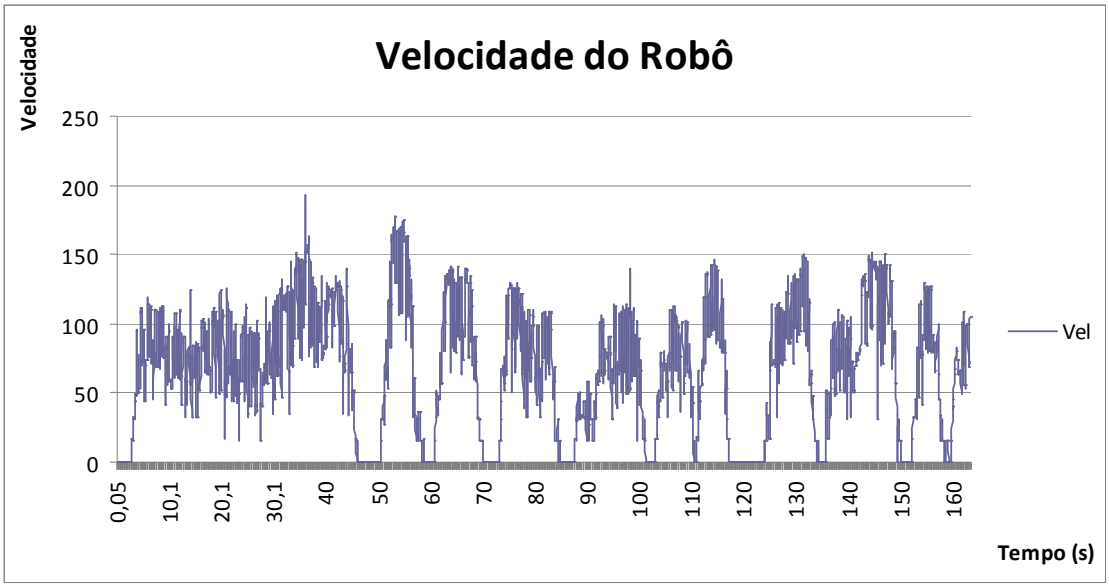


**Gráfico 5 Velocidade do Robô**

Fazendo uma comparação directa com o Gráfico 4, verifica-se que quando a velocidade se anula, os valores de corrente para ambos os motores também são baixos.

## 7.4. GPS

Registaram-se ainda, durante cinco minutos os valores recebidos pelo receptor de GPS.

Apresenta-se no Gráfico 6 a relação entre as coordenadas (latitude e longitude) recebidas pelo receptor com o robô parado.

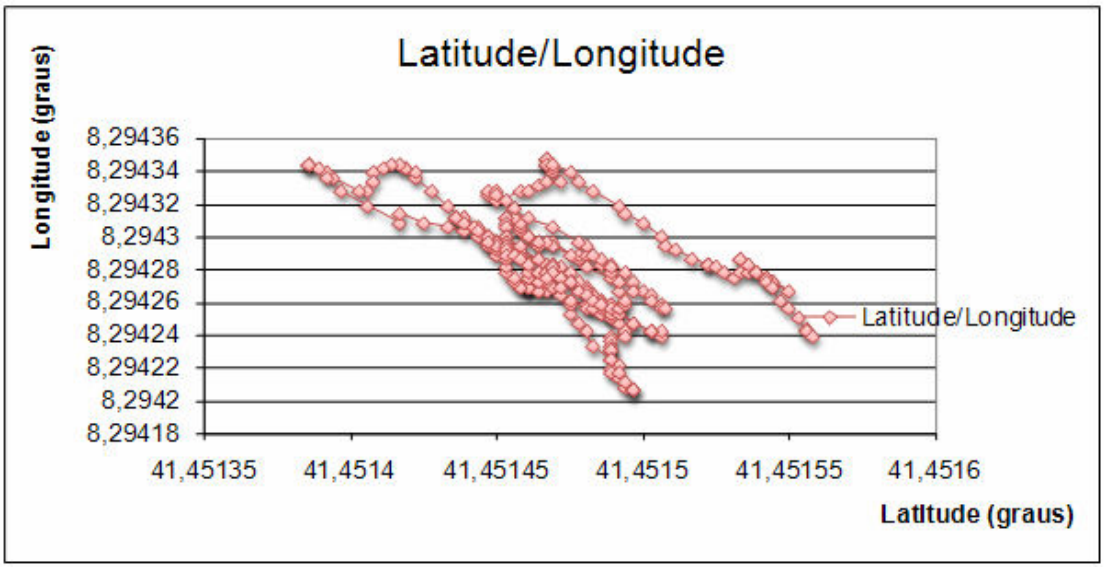


**Gráfico 6 Latitude / Longitude**

Analisando o gráfico, verifica-se a variação das coordenadas, embora exista um pequeno local no centro do gráfico onde os valores estão mais aproximados.

Foram ainda realizados outros testes às coordenadas recebidas do receptor GPS.

Realizou-se a transformação das coordenadas Polares P → (latitude; longitude) para cartesianas P → (X;Y;Z);

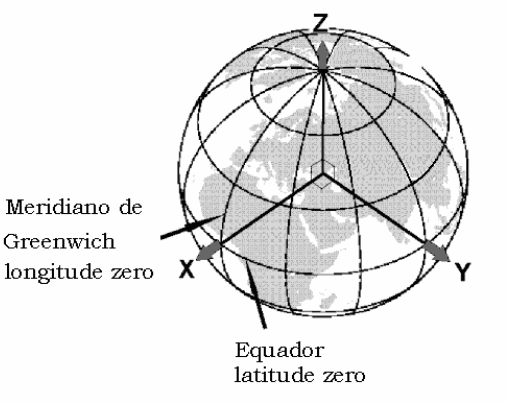


**Figura 60 Coordenadas Cartesianas**

Onde θ é a latitude, φ a longitude e r o valor aproximado do raio da terra (6378Km).

$$X = r \cos \boldsymbol{\theta} \cos \boldsymbol{\varphi}$$
$$Y = r \cos \boldsymbol{\theta} \operatorname{sen} \boldsymbol{\varphi}$$
$$Z = r \operatorname{sen} \boldsymbol{\theta}$$

Tomando como referencia o ponto (4730,4;689,6;4222,135), realizando o cálculo da distância ao referido ponto, temos:

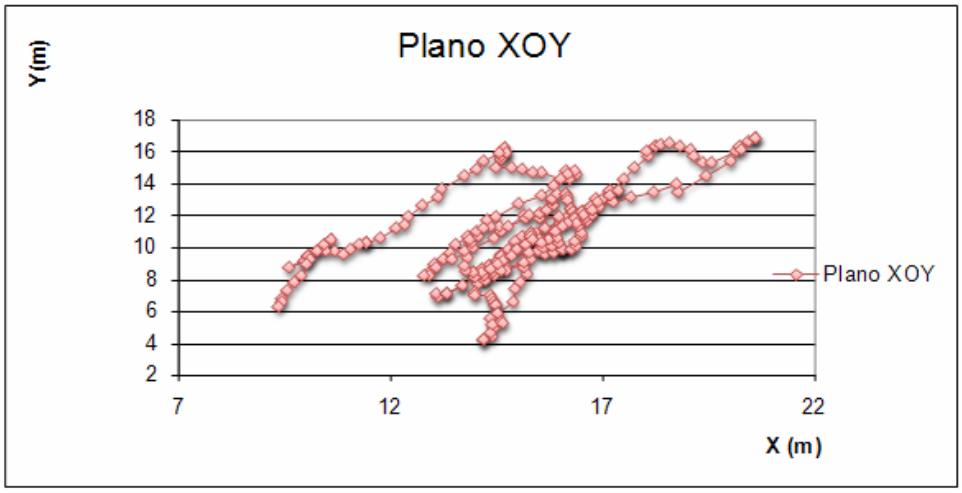


**Gráfico 7 Variação coordenadas GPS no plano XOY**

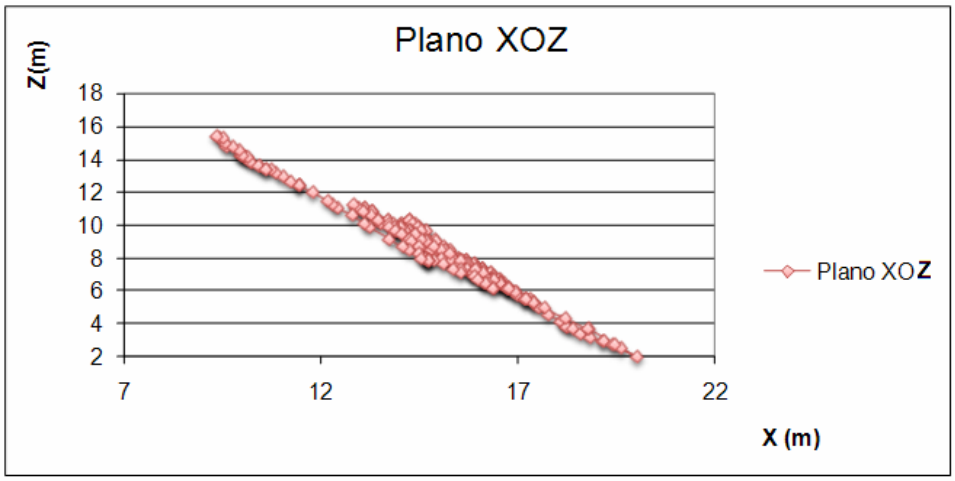


**Gráfico 8 Variação coordenadas GPS no plano XOZ**

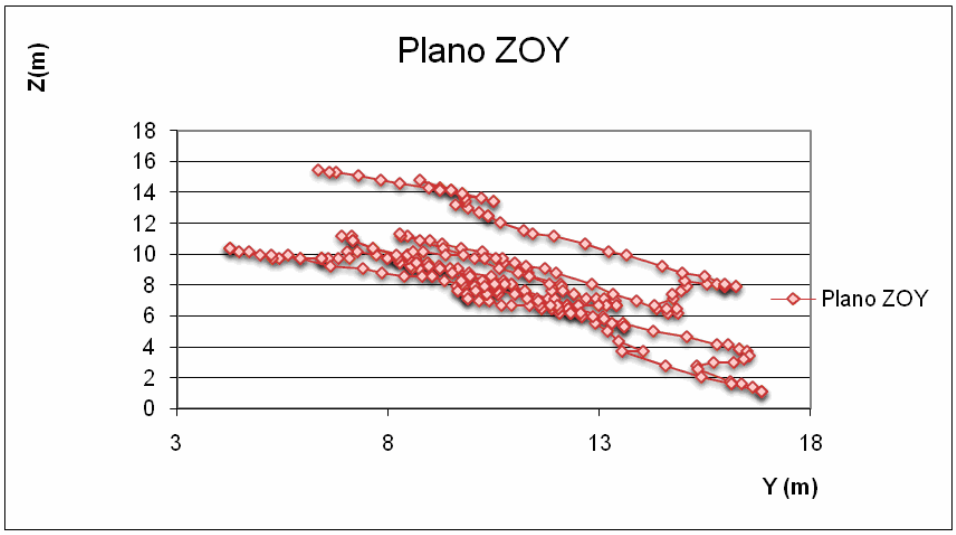


**Gráfico 9 Variação coordenadas GPS no plano ZOY**

**Tabela 18 Variação de X, Y e Z**

| | | |
|---|---|---|
| Para o eixo X | | |
| | Valor Médio | 14,983321499843 |
| | Máximo | 20,614909953110 |
| | Mínimo | 9,336532833913 |
| Para o eixo Y | | |
| | Valor Médio | 10,844453348676 |
| | Máximo | 16,837871705320 |
| | Mínimo | 4,264434307061 |
| Para o eixo Z | | |
| | Valor Médio | 8,409307605149 |
| | Máximo | 15,471498573788 |
| | Mínimo | 1,120846742197 |

Registaram-se ainda um conjunto de valores de distâncias ao alvo, estando o robô e o alvo parados. Apresentam-se no Gráfico 10 o valor da distância ao ponto dt e as suas componentes dx e dy.

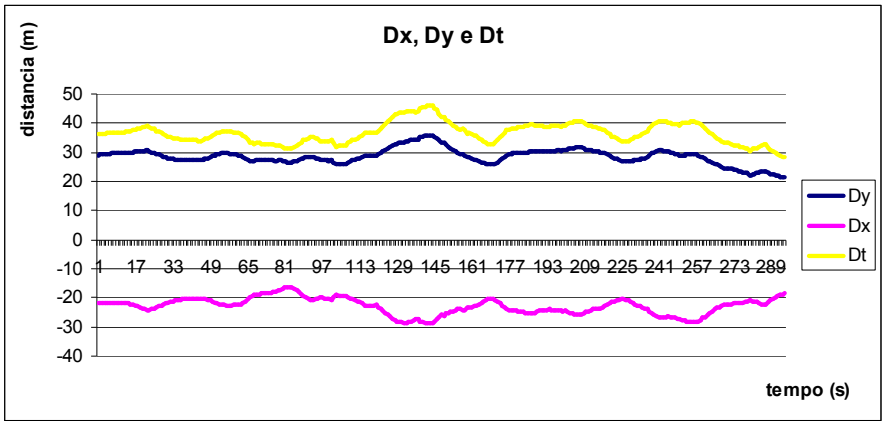


**Gráfico 10 Dx, Dy e Dt**

Apresentam-se na Tabela 19 o valor médio, máximo e mínimo da distância do robô ao alvo *dt* e das suas componentes *dx* e *dt*.

**Tabela 19 Variação da distância**

| Dx (m) | | Dy(m) | | Dt (m) | |
|---|---|---|---|---|---|
| Valor Médio | -22,844453348734 | Valor Médio | 28,590692394203 | Valor Médio | 36,633106222805 |
| Máximo | -16,264434307118 | Máximo | 35,879153257156 | Máximo | 46,031907226904 |
| Mínimo | -28,837871705377 | Mínimo | 21,528501425564 | Mínimo | 28,281629517709 |

Analisando a Tabela 19, verifica-se que a distância no máximo varia 18m

Tendo como base o Gráfico 10, realizou-se o Gráfico 11, onde se observam os valores que a variação da distância causa na direcção desejada do robô.

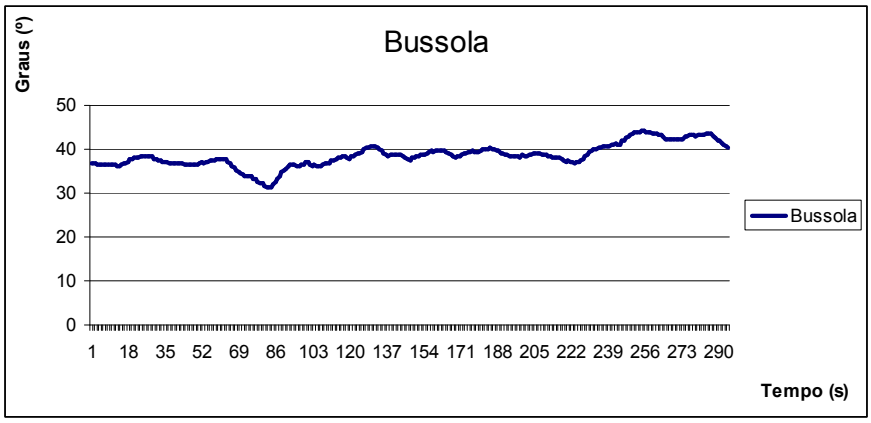


**Gráfico 11 Bússola**

Apresentam-se na Tabela 20 o valor médio, máximo e mínimo de direcção do robô, durante o teste realizado.

**Tabela 20 Variação dos valores da bússola calculada**

| Bússola | |
|---|---|
| Valor Médio | 38,566389840247 |
| Máximo | 44,104346929306 |
| Mínimo | 31,426267420948 |

Usando a trama $PGRME, durante o tempo das leituras anteriores, registaram-se quais os valores estimados de erro. Apresentam-se na Gráfico 12.

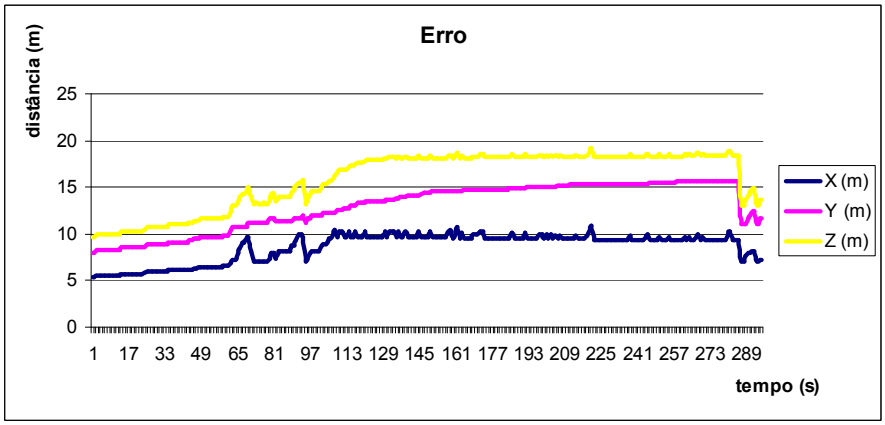


**Gráfico 12 Erro estimado**

Realizaram-se ainda outros registos, entre os quais se destacam o número de satélites com que o receptor GPS conseguiu ligar, apresentado no Gráfico 13.

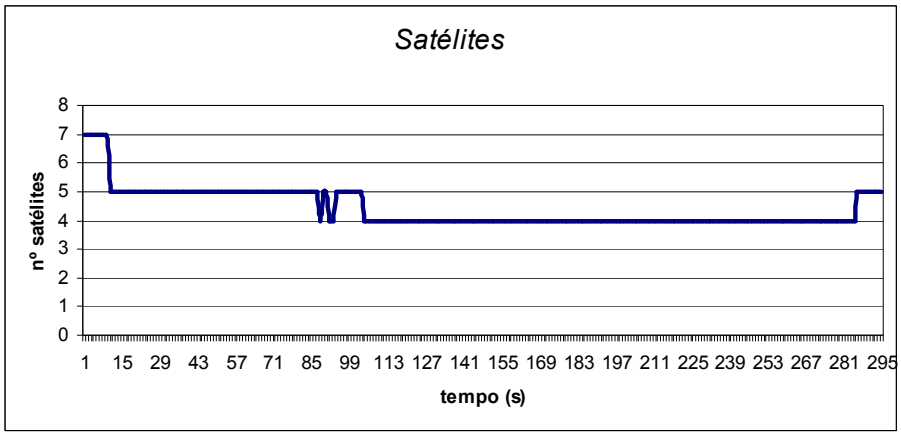


**Gráfico 13 Número de Satélites**

Verificou-se que o número de satélites presente no horizonte varia entre 7 e 4. Note-se que 4 é o número mínimo para a posição obtida seja válida, ou seja, três para o posicionamento e outro para acertar os relógios dos satélites e do receptor GPS.

Comparando ainda com o gráfico anterior, do erro, verifica-se que quando o número de satélites no horizonte diminui, o valor de erro estimado aumenta.

# 8. Discussão de Resultados

A aplicação desenvolvida é extremamente útil, atendendo ao número de visitas e contactos existentes, desde que foi apresentada na comunicação social.

A inovação, autonomia dada ao sistema submeteu-se á patente portuguesa nº 103807, com registo efectuado em 13 de Agosto de 2007 com a designação "*Sistema de recolha de bolas de golfe totalmente autónomo ou remotamente operado*".

Trata-se de um sistema bastante complexo que envolve várias áreas, mecânica, programação, electrónica, comunicação, controlo, matemática, entre muitos outros.

As placas de controlo dos motores utilizadas, revelaram-se bastante versáteis e eficazes. De notar a extrema facilidade com que se controla um motor e o número de variáveis que podemos ter acesso.

Optou-se por utilizar estas placas MD03, uma vez que são sistemas comercializados e apresentam uma eficácia e rendimentos bastante superiores quando comparados com placas desenvolvidas no meio académico.

O receptor de GPS utilizado não permite o uso do sistema de correcção posicional EGNOS. Somente permite usar o WAS, o que ainda leva a um erro maior, porque as estações de referência estão colocadas no continente dos Estados Unidos da América.

Devido ao erro excessivo do GPS, somente está definida uma área de funcionamento e distância de segurança.

Porém encontra-se em desenvolvimento uma solução diferencial, com posicionamento relativo a uma base fixa.

É fundamental ter o receptor GPS para realizar o posicionamento do robô.

Por realizar leituras do campo magnético da Terra, a bússola é um sensor muito susceptível a interferências externas, como estruturas metálicas, necessitando de uma validação prévia aquando da escolha da sua posição.

O norte magnético não coincide com o norte geográfico. Esta diferença denomina-se de declinação magnética e o seu valor em Portugal é de 9º. Deste modo, é necessário corrigir o valor lido pela bússola.

Valores elevados de humidade influenciam as medições realizadas pelos dispositivos de ultra-som, assim como o correcto funcionamento dos restantes componentes.

Para saber qual a inclinação e aceleração actual do robô é necessário realizar uma monitorização permanente do estado dos portos do acelerómetro para realizar o cálculo do *duty-cycle*.

A humidade é normalmente difícil de medir. Alguns dispositivos para medição de humidade exigem circuitos que façam a conversão analógico-digital, onde são necessários amplificadores operacionais, osciladores, compensações, ajustes de temperatura para calcular o "dew-point" – ponto de orvalho, calibração conhecendo um valor referência de humidade, resposta em tempo útil e protecção contra as condições climatéricas. São necessários dispositivos algo complexos.

Por exemplo num dia de chuva em que o valor previsto era de 92%, o valor obtido no sensor foi de 82%. Note-se que o sensor está colocado dentro de uma caixa

Note-se que, quando todo o sistema é ligado, inclusive o de ventilação o valor de Humidade diminui.

O uso de *threads* permitiu a execução simultânea de vários processos.

# 9. Conclusões e trabalho futuro

## 9.1. Conclusões e trabalho futuro

O robô está funcional e em utilização no campo de treinos de golfe da Universidade do Minho.

Os principais objectivos deste trabalho formam concluídos havendo ainda muito a fazer para tornar este robô num produto comercial.

Temos que a visão é um ponto crítico e fundamental, porque permite ao robô localizar as bolas, para corrigir a sua trajectória.

O grupo de sensores deste robô permite que realize a tarefa para o qual foi construído.

A solução implementada neste trabalho deu origem a patente portuguesa nº 103807, com registo efectuado em 13 de Agosto de 2007 com a designação "*Sistema de recolha de bolas de golfe totalmente autónomo ou remotamente operado*", Na qual se descreve em pormenor a plataforma móvel autónoma de recolha de bolas de golfe.

Deu ainda origem a um paper submetido à revista do Departamento de Electrónica Industrial da Universidade do Minho "*Plataforma móvel para recolha autónoma de bolas de golfe*".

## 9.2. Trabalho futuro

Como o robô se encontra na fase de protótipo, existe ainda muito trabalho a ser realizado.

Criação de uma nova estrutura mecânica mais leve, alterar o modo de acoplamento dos motores e testar a necessidade de usar reduções para diminuir o binário pedido aos motores.

Optimização do controlo da velocidade dos motores, porque o atrito com a irregularidade do terreno, condições atmosféricas, número de bolas recolhidas, etc.

Utilizar os valores fornecidos pelo acelerómetro para melhorar o controlo da velocidade dos motores.

Optimização do sistema de posicionamento, usando um posicionamento relativo a uma base fixa, reduzindo assim o erro.

Note-se que é necessário adquirir dispositivos ultra-som que suportem valores de humidade consideráveis, ou seja, adequados a utilização no exterior, tais como os presentes nos automóveis. É necessário realizar a aquisição e tratamento do sinal, com vista a tornar possível que o robô evite obstáculos autonomamente.

Podem-se ainda adoptar outro tipo de sensores tais como infravermelhos para realizar a detecção de obstáculos.

Optimizar o sistema de posicionamento.

Realização de testes com sensores magnéticos e fio no solo, para limitar com segurança o espaço de trabalho do robô e entrar na estação de descarga de bolas.

Realização de carga automática das baterias.

Quando existirem valores de humidade elevados, permitir que o sistema se desligue.

Definir novas áreas de acção do robô, com distâncias de segurança em relação aos obstáculos, assim como rotas de viagem do robô no terreno.

# 10. Referências


[1] Gonçalo Prates NAVSTAR GPS -SISTEMA DE POSICIONAMENTO GLOBAL, Página na Internet http://w3.ualg.pt/~gprates/navstar_gps.pdf

[2] How GPS Works: WAAS and EGNOS, Página na Internet http://www.kowoma.de/en/gps/waas_egnos.htm

[3] Sérgio Alves, A GEOMETRIA DO GLOBO TERRESTRE, Página na Internet http://www.ime.unicamp.br/~eliane/ma241/trabalhos/globo.pdf

[4] Sirf NMEA Reference Manual Página na Internet http://www.sparkfun.com/datasheets/GPS/NMEA%20Reference%20Manual1.pdf

[5] SiRF Binary Protocol Reference Manual, Página na Internet http://www.fastrax.fi/showfile.cfm?guid=d27f8ed2-7944-4710-851e-bebb5c07ad99

[6] Maria Isabel Ribeiro Uma Viagem ao Mundo dos Robots, Página na Internet http://users.isr.ist.utl.pt/~mir/pub/ViagemRobots-IsabelRibeiro05.pdf

[7] Portugal Golf Equipamento e Acessórios, Página na Internet http://www.portugalgolf.pt/golfe/golf_equip_aces_3.htm

[8] Playgolf – O seu Caddy online, página na Internet http://www.playgolf.pt/content.php?section=golfmania

[9] FOX Board, a complete Linux system in just 66 x 72 mm. Consultado em Maio de 2007, Página na Internet http://www.acmesystems.it/?id=4

[10] Web-Compiler, Página na Internet http://www.acmesystems.it/?id=200

[11] Install the SDK on Linux, Página na Internet http://www.acmesystems.it/?id=714

[12] How compile a C Application, Página na Internet http://www.acmesystems.it/?id=711

[13] What is ZigBee®?. Consultado em Junho de 2007, Página na Internet http://www.maxstream.net/wireless/zigbee.php

[14] Memsic 2125 Dual-axis Accelerometer. Consultado em Junho de 2007, Página na Internet

http://www.parallax.com/detail.asp?product_id=28017
[15] Accelerometer - Getting Started. Consultado em Junho de 2007, Página na Internet http://www.parallax.com/dl/docs/prod/compshop/SICMemsicTut.pdf
[16] Quick Assembly Two and Three Channel Optical Encoders Technical Data. Consultado em Junho de 2007, Página na Internet http://pdf1.alldatasheet.com/datasheet-pdf/view/88208/HP/HEDS5540.html
[17] MD03 Technical Data. Consultado em Junho de 2007 Página na Internet http://www.robot-electronics.co.uk/htm/md03tech.htm MD03
[18] SRF235 Ultrasonic range finder Technical Specification. Consultado em Setembro de 2007, Página na Internet http://www.robot-electronics.co.uk/htm/srf235tech.htm
[19] Isaac Asimov Página na Internet http://www.asimovonline.com/
[20] "Robotics Industries Association", Página na Internet http://www.roboticsonline.com/
[21] Peyman Gohari" Digital Control Algorithms and Their Implementation" Página na Internet http://users.encs.concordia.ca/~gohari/ELEC6061/Files/9.pdf
[22] Sergio Tanzilli *"Using Serial Ports in C"* Página na internet http://www.acmesystems.it/?id=35
[23] "*FOXZB a ZigBee/RS485 add-on board*" Página na internet http://www.acmesystems.it/?id=42
[24] *Data Radio Modems and Radio Modules* Página na internet http://www.maxstream.net/
[25] Ballpicker Página na internet http://www.bigmow.biz/pages/ballpicker.htm
*[26]* A. Bernardino, J. Miranda Lemos, *"Modelação, Identificação e Controlo Digital"* Página na internet http://users.isr.ist.utl.pt/~alex/micd0506/docs.htm
[27] Apontamentos da disciplina de Controlo Digital no ano lectivo 2005/2006 com a Professora Estela Bicho
[28] Sistema Esférico de Coordenadas Página na Internet http://pt.wikipedia.org/wiki/Sistema_esf%C3%A9rico_de_coordenadas

[29] Federação Portuguesa de Orientação - Bússola
Página na Internet
http://www.fpo.pt/o_que_e/material_bussola.html
[30] Invivo – Museu da vida – COC – FioCruz
Página na Internet
http://www.museudavida.fiocruz.br/publique/cgi/cgilua.exe/sys/start.htm?sid=7&infoid=798
*[31]* Sergio Tanzilli, *"Using bit banging method in user space to add an I2C bus"*
http://www.acmesystems.it/?id=10
[32] I2C Application Note
Página na internet
http://www.nxp.com/acrobat_download/applicationnotes/AN10216_1.pdf
[33] Axis Communications – System-on-Chips & Device Servers
Página na Internet
http://www.axis.com/products/dev/index.htm
[34] Fox Pinout
Página na Internet
http://www.acmesystems.it/?id=18
[35] Using the I/O lines in C
Página na Internet
http://www.acmesystems.it/?id=22
[36] Sensirion Temperature/Humidity Sensor
Página na Internet
http://www.parallax.com/Store/Sensors/TemperatureHumidity/tabid/174/CategoryID/49/List/0/Level/a/ProductID/94/Default.aspx?SortField=ProductName%2CProductName
[37] GPS 18 TECHNICAL SPECIFICATIONS
Página na Internet
http://www8.garmin.com/manuals/425_TechnicalSpecification.pdf
[38] Robot Compass Module Technical Specification. Consultado em Setembro de 2007
Página na Internet
http://www.robot-electronics.co.uk/htm/cmps3doc.shtml
[39] Documentação técnica do conversor CC-CC PL5S12-C
[40] Documentação técnica do conversor CC-CC R785.0-0.5

# 11. Bibliografia


Segantine, Paulo Cesar Lima, GPS Sistema de Posicionamento Global, Suprema Gráfica Editora, 2005
*"O panorama do golfe em Portugal"*,

Página na Internet

http://if.fc.ul.pt/golfe/com/icep.htm

Neil H. Kussat, C. David Chadwell, Member, IEEE, and Richard Zimmerman, Member, IEEE
*"Absolute Positioning of an Autonomous Underwater Vehicle Using GPS and Acoustic Measurements"*

IEEE JOURNAL OF OCEANIC ENGINEERING, VOL. 30, NO. 1, JANUARY 2005
Página na Internet

http://ieeexplore.ieee.org/iel5/48/30931/01435583.pdf?arnumber=1435583

Tese de Justo Emilio Alvarez Jácobo
*"Desenvolvimento de um Robô Autônomo Móvel Versátil utilizando Arquitetura Subsumption"*

Página na Internet

http://www.fem.unicamp.br/~lva/pdf/Pablo/TeseJustoEmilioAlvarez2001.pdf

Tese de Cleber Zorzi
*"CONTROLADOR PID DIGITAL DE VELOCIDADE DE UM MOTOR DE CORRENTE CONTÍNUA"*

J. Norberto Pires
*"Robótica – Das máquinas gregas à moderna Robótica Industrial"*

Página na Internet

http://robotics.dem.uc.pt/norberto/

Paper de
Guilherme de Lima Ottoni e Walter Fetter Lagesy
*"NAVEGAÇÃO DE ROBÔS MÓVEIS EM AMBIENTES DESCONHECIDOS UTILIZANDO SONARES DE ULTRA-SOM"*

Página na Internet

http://liberty.princeton.edu/Publications/ca14_robot.pdf

UnilesteMG – Curso de Especialização em Controle de Processos Industriais
Luiz Carlos Figueiredo
*"Controladores e Ações de Controle"*

Página na Internet

http://professores.unilestemg.br/~figueiredo/tcpi_c3.pdf

# 12. Anexos

Em anexo, colocou-se o código realizado para o robô.

```
1. /*
2.       Software realizado para o Robo do Golf no ano 2006/2007
3. */
4. #include <string.h>
5. #include <unistd.h>
6. #include <sys/ioctl.h>
7. #include <fcntl.h>
8. #include <time.h>
9. #include <asm/etraxgpio.h>
10. #include <sys/time.h>
11. #include <stdlib.h>
12. #include <math.h>
13. #include <tgmath.h>
14. #include <sys/un.h>
15. #include <unistd.h>
16. #include <sys/poll.h>
17. #include <errno.h>
18. #include <sys/types.h>
19. #include <sys/stat.h>
20. #include <stdarg.h>
21. #include <termios.h>
22. #include <pthread.h>
23. #include <sys/socket.h>
24. #include <netinet/in.h>
25. #include <arpa/inet.h>
26. #include <netdb.h>
27. /*******************************************/
28. //GPS
29. int dirG = 0;
30. typedef struct
31. {
32.       double longitude;
33.       double latitude;
34. }TRAMA_GPS;
35. TRAMA_GPS gps;
```

```
36. char nsat[10];
37. char tempGPS[10];
38. float erroX,erroY,erroZ,velGPS1;
39. char velGPS[10];
40. int nsatelites;  //atoi
41. float temperatura_GPS; //atof
42. int Flag_comunica1 = 1;
43. #define IP_SERVER "192.168.0.91"
44. #define PORT_SERVER 50000
45. //Barramento i2c
46. int i2c_fd;
47. #define I2C_DATA_LINE   1<<24 // Linha de dados do I2C
48. #define I2C_CLOCK_LINE  1<<25 // Linha de Clock do I2C
49. //sensor de Humidade
50. typedef union
51. {
52.       char i;
53.       float f;
54. } value_;
55. enum {TEMP,HUMI};
56. #define noACK     0
57. #define ACK       1
58. #define  STATUS_REG_W         0x06        //000 00110
59. #define  STATUS_REG_R         0x07        //000 00111
60. #define  MEASURE_TEMP         0x03        //000 00011
61. #define  MEASURE_HUMI         0x05        //000 00101
62. #define  RESET                0x1e  //000 11110
63. float temp, humi, dew_point;
64. //Encoders
65. float velocidade_ED = 0.00;  //em m/s
66. int velocidadeD = 0;
67. float velocidade_EE = 0.00;  //em m/s
68. int velocidadeE = 0;
69. float vel_linear = 0.00;
70. int iomaskE=1<<8;
71. int iomaskD=1<<3;
72. float KPE=1.00;
73. float KIE=0.05;
74. float KPD=1.00;
75. float KID=0.05;
76. float veld1D=0.0,velocidade_D=0.00;
```

```
77. float veld1E=0.0,velocidade_E=0.00;
78. int velE=0,velD=0;
79. int flagcontrolo =0 ;   //Activar Desactivar controlo PI
80. #define MOTOR_DIR 0xB2        // Identificacao da md03 da direita
81. #define MOTOR_ESQ 0xB0        // Identificacao da md03 da esuqerda
82. int ACEL=200;                 // Valor de aceleracao a aplicar
83. int Flag_stop_joystic=0;
84. int Flag_comunica=1;
85. int alarme=0;
86. int Flag_cala_t=1;
87. int Flag_Notas=1;
88. unsigned int TEMPE=0;
89. unsigned int TEMPD=0;
90. float ID=0.0;
91. float IE=0.0;
92. float te=0.00;
93. float td=0.00;
94. unsigned int Bussola=0;
95. //Acelerometro
96. struct timeval tv;      //EXTRUTURA DO ACELAROMETRO
97. float Tilt=0.0,Pan=0.0;
98. int fdPin;
99. int iomaskX=1<<2; //Identificar o pino IG2 PINO-->29
100.      int iomaskY=1<<13;
101.      int aceX=1;
102.      int aceY=1;
103.      float vx[100];
104.      float vy[100];
105.      float gY=0.0,gX=0.0;
106.      //JOYSTIC
107.      int fd;
108.      //Ultrasom
109.      int ob=0;
110.      int sonar=0;
111.      int vel=0,dir=0;
112.      int a_correr=1;
113.      int fdGPS = 0;
114.      int print_v = 1;
115.      int ultrasons=0;
116.      int GPS = 1;
117.      int contDY=0;
```

```
118.      float Bateria=0.00;
119.      float Bat1=0.00;
120.      float Bat2=0.00;
121.      float X1=0.00;
122.      float Y1=0.00;
123.      float X2=0.00;
124.      float Y2=0.00;
125.      // leitura de teclas
126.      char mygetch( )
127.      {
128.      struct termios oldt,newt;
129.      char ch;
130.            tcgetattr( STDIN_FILENO, &oldt );
131.            newt = oldt;
132.            newt.c_lflag &= ~( ICANON | ECHO );
133.            tcsetattr( STDIN_FILENO, TCSANOW, &newt );
134.            ch = getchar();
135.            tcsetattr( STDIN_FILENO, TCSANOW, &oldt );
136.      return ch;
137.      }
138.      void print_menu(void)
139.      {
140.      printf("    GOLF\n\n");
141.      printf("Velocidade  - (q++)--(a--)\n");
142.      printf("Direccao    - (o++)--(p--)\n");
143.      printf("Aceleraccao - (z++)--(x--)\n");
144.      printf("' ' - PARAGEM DE EMERGENCIA\n");
145.      printf("ESC - SAIR\n");
146.      printf("\n\n");
147.      }
148.      void error(char *msg)
149.      {
150.          perror(msg);
151.          exit(0);
152.      }
153.      void enviar_trama(char* msg, int number_bytes)
154.      {
155.      int sock, length, n;
156.      struct sockaddr_in server;
157.      struct hostent *hp;
158.            sock= socket(AF_INET, SOCK_DGRAM, 0);
```

```
159.            if (sock < 0)
160.                  error("socket");
161.            server.sin_family = AF_INET;
162.            hp = gethostbyname("192.168.0.91");  //"IP_SERVER"
163.            if (hp == 0)
164.                  error("Unknown host");
165.            bcopy((char *)hp->h_addr, (char *)&server.sin_addr, hp-
>h_length);
166.            server.sin_port = htons(50000);     //PORT_SERVER
167.            length=sizeof(struct sockaddr_in);
168.            n=sendto(sock,msg,1024,0,(struct sockaddr
*)&server,length);
169.            if (n < 0)
170.                  error("Sendto");
171.      close(sock);
172.      }
173.      //IMPRESSÃƒO DE VARIAVEIS UTEIS
174.      void print_variaveis(int velocidade, int direccao)
175.      {
176.      char info[1024];
177.            printf("\t\t\t\tVel=%4d Dir=%4d ESQ: I=%.2f DIR: I=%.2f
Bus=%d P=%.2f T=%.2f S= %d T:%5.1fC   H:%5.1f%%
\r",velocidade,direccao,IE,ID,Bussola,Pan,Tilt,sonar,temp,humi);
178.      nsatelites = atoi(nsat);
179.      temperatura_GPS = atof(tempGPS);
180.            if(Flag_comunica == 0)
181.            {
182.
   sprintf(info,"GOLF,%f,%f,%f,%f,%d,%f,%lf,%lf,%d,%f,%f,%f,%f,%f",vel_li
near,
humi,Pan,Tilt,Bussola,temp,gps.longitude,gps.latitude,nsatelites,temperatu
ra_GPS,erroX,erroY,erroZ,velGPS1);
183.                  enviar_trama(info, 1024);//((char *)&trama)
184.            }
185.      }
186.      //LÃª ficheiro de configuraÃ§Ã£o
187.      int le_config(char *nomeficheiro)
188.      {
189.      FILE *fp2;
190.      char caracter;
191.      char str1[80], str2[80];
```

```
192.           if ((fp2=fopen(nomeficheiro, "r"))==NULL)
193.           {
194.                 printf("Erro na leitura do ficheiro %s.\n",
nomeficheiro);
195.                 return 0;
196.           }
197.           while (fscanf(fp2,"%c",&caracter), !feof(fp2))
198.           {
199.                 if (caracter=='#')
200.                       fscanf(fp2,"%*[^\n]%*c");
201.                 else
202.                       if (caracter!='\n')
203.                       {
204.                             fseek(fp2,-1,1);
205.                             fscanf(fp2, "%[^= ]=%[^\n]%*c", str1,
str2);
206.                             if (!strcmp(str1, "KPD"))
   sscanf(str2, "%f", &KPD);
207.                             if (!strcmp(str1, "KID"))
   sscanf(str2, "%f", &KID);
208.                             if (!strcmp(str1, "KPE"))
   sscanf(str2, "%f", &KPE);
209.                             if (!strcmp(str1, "KIE"))
   sscanf(str2, "%f", &KIE);
210.                       }
211.           }
212.           fclose(fp2);
213.           printf("\n\n\t\t\t\tIP_SERVER= %s PORT_SERVER=%d KPD= %.2f
KID= %.2f",IP_SERVER, PORT_SERVER, KPD,KID);
214.     return 1;
215.     }
216.     void escreve_ficheiro(char *nomeficheiro, char *val1)
217.     {
218.     FILE *fpe;
219.           fpe=fopen(nomeficheiro, "a");
220.             fprintf(fpe, "\n%s",val1);
221.     fclose(fpe);
222.     }
223.     //Limitar velocidades e direccao
224.     void limita_vel(int dir1, int vel1) //Calculo das velocidades
desejadas para aplicar no controlo de cada motor
```

```
225.     {
226.           velE = vel1 - (dir1/2);
227.           velD = vel1 + (dir1/2);
228.     //virar para a esquerda
229.           if(vel1 > 0 && dir1 > 0 && velE < 0)
230.           {
231.                 velD += abs(velE);
232.                 velE = 0;
233.           }
234.      //virar para a direita
235.           if(vel1 > 0 && dir1 < 0 && velD < 0)
236.           {
237.                 velE +=abs(velD);
238.                 velD = 0;
239.           }
240.           if (vel1 > 255) vel1 = 255;
241.           if (vel1 < -255) vel1 = -255;
242.           if (velE > 255) velE = 255;
243.           if (velD > 255) velD = 255;
244.           if (velE < -255) velE = -255;
245.           if (velD < -255) velD = -255;
246.     }
247.     // Porta serie
248.     int initport(int fd1)
249.     {
250.     struct termios options;
251.           // Get the current options for the port...
252.           tcgetattr(fd1, &options);
253.           // Set the baud rates to 9600...
254.           cfsetispeed(&options, B9600);
255.           cfsetospeed(&options, B9600);
256.           // Enable the receiver and set local mode...
257.           options.c_cflag |= (CLOCAL | CREAD);
258.           options.c_cflag &= ~PARENB;
259.           options.c_cflag &= ~CSTOPB;
260.           options.c_cflag &= ~CSIZE;
261.           options.c_cflag |= CS8;
262.           // Set the new options for the port...
263.           tcsetattr(fd1, TCSANOW, &options);
264.     return 1;
265.     }
```

```
266.    int writeport(int fd1, char *chars)
267.    {
268.    int len = strlen(chars);
269.          //chars[len] = 0x0d; // stick a <CR> after the command
270.          //chars[len+1] = 0x00; // terminate the string properly
271.          printf("\n\nSERIAL PORT: caracteres %d\r", len);
272.          int n = write(fd1, chars, strlen(chars));
273.          if (n < 0) {
274.                fputs("SERIAL PORT: write failed!\r", stderr);
275.                return 0;
276.          }
277.    return 1;

278.    }
279.    int writeport1(int fd1, char *chars)
280.    {
281.    int len = strlen(chars);
282.    chars[len] = 0x0d; // stick a <CR> after the command
283.    chars[len+1] = 0x00; // terminate the string properly
284.          int n = write(fd1, chars, strlen(chars));
285.          if (n < 0) {
286.                printf("ERROR %d\r",__LINE__);
287.          return 0;
288.          }
289.    return 1;

290.    }
291.    int readport(int fd1, char *result)
292.    {
293.    int iIn = read(fd1, result, 50);
294.          result[iIn-1] = 0x00;
295.          if (iIn < 0)
296.          {
297.                if (errno == EAGAIN)
298.                {
299.                      printf("SERIAL EAGAIN ERROR\r");
300.                return 0;
301.                }
302.                else
303.                {
```

```
304.                        printf("SERIAL read error %d %s\r", errno,
strerror(errno));
305.                  return 0;
306.                  }
307.            }
308.            return 1;
309.      }
310.      void CloseJAdrPort()
311.      {
312.            // you may want to restore the saved port attributes
313.         if (fd > 0)
314.          {
315.              close(fd);
316.          } // end if
317.      } // end CloseAdrPort
318.      int init_comandoAT(int fd1)
319.      {
320.      char sCmd[254]={"+++"};
321.            usleep(1000000);
322.            if (!writeport(fd1, sCmd)) {
323.                  //printf("Comando AT: write failed\r");
324.                  printf("ERROR  %d\r",__LINE__);
325.                  close(fd1);
326.                  return 0;
327.            }
328.            usleep(1500000);
329.            char sResult[254];
330.            fcntl(fd1, F_SETFL, FNDELAY); // don't block serial read
331.            if (!readport(fd1,sResult)) {
332.                  printf("ERROR  %d\r",__LINE__);
333.                  return 0;
334.            }
335.            return 1;
336.      }
337.      int comandoAT(int fd1, char *chars)
338.      {
339.            if (!writeport1(fd1, chars)) {
340.                  //printf("Comando AT: write failed\r");
341.                  printf("ERROR  %d\r",__LINE__);
342.                  close(fd1);
343.                  return 0;
```

```
344.           }
345.           printf("Comando AT: written:%s\r", chars);
346.           usleep(1500);
347.           //usleep(150000);
348.           char sResult1[254];
349.           fcntl(fd1, F_SETFL, FNDELAY); // don't block serial read
350.           if (!readport(fd1,sResult1)) {
351.                 //printf("Comando AT: read failed\r");
352.                 printf("ERROR  %d\r",__LINE__);
353.                 return 0;
354.           }
355.     //    printf("Comando AT: readport=%s\r", sResult1);
356.     return 1;
357.     }
358.     int getbaud(int fd1)
359.     {
360.     struct termios termAttr;
361.     int inputSpeed = -1;
362.           speed_t baudRate;
363.           tcgetattr(fd1, &termAttr);
364.           /* Get the input speed. */
365.           baudRate = cfgetispeed(&termAttr);
366.           switch (baudRate)
367.            {
368.                 case B0:      inputSpeed = 0; break;
369.                 case B50:     inputSpeed = 50; break;
370.                 case B110:    inputSpeed = 110; break;
371.                 case B134:    inputSpeed = 134; break;
372.                 case B150:    inputSpeed = 150; break;
373.                 case B200:    inputSpeed = 200; break;
374.                 case B300:    inputSpeed = 300; break;
375.                 case B600:    inputSpeed = 600; break;
376.                 case B1200:   inputSpeed = 1200; break;
377.                 case B1800:   inputSpeed = 1800; break;
378.                 case B2400:   inputSpeed = 2400; break;
379.                 case B4800:   inputSpeed = 4800; break;
380.                 case B9600:   inputSpeed = 9600; break;
381.                 case B19200:  inputSpeed = 19200; break;
382.                 case B38400:  inputSpeed = 38400; break;
383.           }
384.           return inputSpeed;
```

```
385.      }
386.      // ROTINAS do I2C
387.      // Get the SDA line state
388.      int i2c_getbit(void)
389.      {
390.      unsigned int value;
391.
392.            value=ioctl(i2c_fd, _IO(ETRAXGPIO_IOCTYPE, IO_READBITS));
393.            if ((value&(I2C_DATA_LINE))==0)
394.                  return 0;
395.            else
396.                  return 1;
397.      }
398.      // Set the SDA line as output
399.      void i2c_dir_out(void)
400.      {
401.      int iomask;
402.            iomask = I2C_DATA_LINE;
403.            ioctl(i2c_fd, _IO(ETRAXGPIO_IOCTYPE, IO_SETGET_OUTPUT),
&iomask);
404.      }
405.      // Set the SDA line as input
406.      void i2c_dir_in(void)
407.      {
408.      int iomask;
409.            iomask = I2C_DATA_LINE;
410.            ioctl(i2c_fd, _IO(ETRAXGPIO_IOCTYPE, IO_SETGET_INPUT),
&iomask);
411.      }
412.      // Set the SDA line state
413.      void i2c_data(int state)
414.      {
415.            if (state==1)
416.            {
417.                  i2c_dir_in();
418.            }
419.            else
420.            {
421.                  i2c_dir_out();
422.                  ioctl(i2c_fd, _IO(ETRAXGPIO_IOCTYPE, IO_CLRBITS),
I2C_DATA_LINE);
```

```
423.            }
424.      }
425.      // Set the SCL line state
426.      void i2c_clk(int state)
427.      {
428.            if (state==1)
429.                  ioctl(i2c_fd, _IO(ETRAXGPIO_IOCTYPE, IO_SETBITS),
I2C_CLOCK_LINE);
430.            else
431.                  ioctl(i2c_fd, _IO(ETRAXGPIO_IOCTYPE, IO_CLRBITS),
I2C_CLOCK_LINE);
432.      }
433.      // Read a byte from I2C bus and send the ack sequence
434.      // Put islast = 1 is this is the last byte to receive from the
slave
435.      unsigned char i2c_inbyte(int islast)
436.      {
437.      unsigned char value = 0;
438.      int bitvalue;
439.      int i;
440.            // Read data byte
441.            i2c_clk(0);
442.            i2c_dir_in();
443.            for (i=0;i<8;i++)
444.            {
445.                  i2c_clk(1);
446.                  bitvalue = i2c_getbit();
447.                  value |= bitvalue;
448.                  if (i<7)
449.                        value <<= 1;
450.                  i2c_clk(0);
451.            }
452.            if (islast==0)
453.            {     // Send Ack if is not the last byte to read
454.                  i2c_dir_out();
455.                  i2c_data(0);
456.                  i2c_clk(1);
457.                  i2c_clk(0);
458.                  i2c_dir_in();
459.            }
460.            else
```

```
461.            {     // Doesn't send Ack if is the last byte to read
462.                  i2c_dir_in();
463.                  i2c_clk(1);
464.                  i2c_clk(0);
465.            }
466.      return value;
467.      }
468.      // Open the GPIOB dev
469.      int i2c_open(void)
470.      {
471.      int iomask;
472.            if ((i2c_fd = open("/dev/gpiog", O_RDWR))<0)
473.            {
474.                  //printf("i2c open error\n");
475.                  printf("ERROR  %d\r",__LINE__);
476.                  return 1;
477.            }
478.            iomask = I2C_CLOCK_LINE;
479.            ioctl(i2c_fd, _IO(ETRAXGPIO_IOCTYPE, IO_SETGET_OUTPUT),
&iomask);
480.            iomask = I2C_DATA_LINE;
481.            ioctl(i2c_fd, _IO(ETRAXGPIO_IOCTYPE, IO_SETGET_INPUT),
&iomask);
482.            ioctl(i2c_fd, _IO(ETRAXGPIO_IOCTYPE, IO_SETBITS),
I2C_DATA_LINE);
483.            i2c_dir_in();
484.            i2c_clk(1);
485.      return i2c_fd;
486.      }
487.      // Close the GPIOB dev
488.      void i2c_close(void)
489.      {
490.            i2c_dir_in(); // TALVEZ NAO SEJA NECESSARIO
491.            close(i2c_fd);
492.      }
493.      // Send a start sequence to I2C bus
494.      void i2c_start(void)
495.      {
496.            i2c_clk(0);
497.            i2c_data(1);
498.            i2c_clk(1);
```

```
499.            i2c_data(0);
500.      }
501.      // Send a stop sequence to I2C bus
502.      void i2c_stop(void)
503.      {
504.            i2c_clk(0);
505.            i2c_data(0);
506.            i2c_clk(1);
507.            i2c_data(1);
508.      }
509.      // Send a byte to the I2C bus and return the ack sequence from
slave
510.      // rtc
511.      //    0 = Nack, 1=Ack
512.      int i2c_outbyte(unsigned char x)
513.      {
514.      int i;
515.      int ack;
516.            i2c_clk(0);
517.            for (i=0;i<8;i++)
518.            {
519.                  if (x & 0x80)
520.                        i2c_data(1);
521.                  else
522.                        i2c_data(0);
523.                  i2c_clk(1);
524.                  i2c_clk(0);
525.                  x <<= 1;
526.            }
527.            i2c_dir_in();
528.            i2c_clk(1);
529.            ack=i2c_getbit();
530.            i2c_clk(0);
531.            if (ack==0)
532.                  return 1;
533.            else
534.                  return 0;
535.      }
536.      //Sensor de Humidade
537.      // writes a byte on the Sensibus and checks the acknowledge
538.      char s_write_byte(unsigned char value)
```

```
539.      {
540.      unsigned char i, errorH=0;
541.
542.            for (i=0x80;i>0;i/=2)   //shift bit for masking
543.            {
544.                  if (i & value)
545.                        i2c_data(1);
546.                  else
547.                        i2c_data(0);
548.                  i2c_clk(1); //usleep(15);
549.                  i2c_clk(0); //usleep(15);
550.            }
551.            i2c_clk(1); //usleep(15);                 //clk #9 for ack
552.            errorH=i2c_getbit();                //check ack (DATA will
be pulled down by SHT11)
553.            i2c_clk(0); //usleep(15);
554.            return errorH;
555.      }
556.      // reads a byte form the Sensibus and gives an acknowledge in
case of "ack=1"
557.      char s_read_byte(unsigned char ack)
558.      {
559.      unsigned char i,valH=0;
560.            for (i=0x80;i>0;i/=2)   //shift bit for masking
561.            {
562.                  i2c_clk(1);
563.                  if (i2c_getbit())
564.                        valH=(valH | i);
565.                  i2c_clk(0);
566.            }
567.            i2c_data(!ack);   //in case of "ack==1" pull down DATA-
Line
568.            i2c_clk(1); //clk #9 for ack
569.            i2c_clk(0);
570.            return valH;
571.      }
572.      void s_transstart(void)
573.      {
574.      i2c_dir_in();
575.            i2c_clk(1);
576.            i2c_data(0);
```

```
577.            i2c_clk(0);
578.            i2c_clk(1);
579.            i2c_data(1);
580.            i2c_clk(0);
581.      }
582.      void s_connectionreset(void)
583.      {
584.      unsigned char i;
585.      i2c_dir_in();
586.            i2c_data(1);
587.            i2c_clk(0);
588.            for(i=0;i<9;i++)
589.            {
590.                  i2c_clk(1);
591.                  i2c_clk(0);
592.            }
593.            s_transstart();
594.      }
595.      // resets the sensor by a softreset
596.      char s_softreset(void)
597.      {
598.      unsigned char errorH=0;
599.            s_connectionreset();          //reset communication
600.            errorH+=s_write_byte(RESET);  //send RESET-command to
sensor
601.            return errorH;                      //error=1 in case of
no response form the sensor
602.      }
603.      // reads the status register with checksum (8-bit)
604.      char s_read_statusreg(unsigned char *p_value, unsigned char
*p_checksum)
605.      {
606.      unsigned char errorH=0;
607.            s_transstart();
   //transmission start
608.            errorH=s_write_byte(STATUS_REG_R);  //send command to
sensor
609.            *p_value=s_read_byte(ACK);                //read status
register (8-bit)
610.            *p_checksum=s_read_byte(noACK);     //read checksum (8-
bit)
```

```
611.            return errorH;                                  //error=1
in case of no response form the sensor
612.      }
613.      // writes the status register with checksum (8-bit)
614.      char s_write_statusreg(unsigned char *p_value)
615.      {
616.      unsigned char errorH=0;
617.            s_transstart();
    //transmission start
618.            errorH+=s_write_byte(STATUS_REG_W);       //send command
to sensor
619.            errorH+=s_write_byte(*p_value);           //send value of
status register
620.            return errorH;                                  //error>=1
in case of no response form the sensor
621.      }
622.      // makes a measurement (humidity/temperature) with checksum
623.      char s_measure(int *p_value, unsigned char *p_checksum, unsigned
char mode)
624.      {
625.      unsigned errorH=0;
626.      unsigned int i;
627.            s_transstart();         //transmission start
628.            switch(mode)            //send command to sensor
629.            {
630.                  case TEMP : errorH+=s_write_byte(MEASURE_TEMP);
//i2c_outbyte(MEASURE_TEMP);
631.                        break;
632.                  case HUMI : errorH+=s_write_byte(MEASURE_HUMI);
//i2c_outbyte(MEASURE_HUMI);
633.                        break;
634.                  default     :
635.                        break;
636.            }
637.            for (i=0;i<65000;i++) //55ms to measure 12bits, 11ms to
8bits
638.            {
639.                  usleep(1);
640.                  if(i2c_getbit()==0)
641.                        break;      //wait until sensor has finished
the measurement
```

```
642.            }
643.            if(i2c_getbit())
644.            {
645.                  printf("\n...TIMEOUT na mediÃ§Ã£o...");
646.                  errorH+=1;  // or timeout (~2 sec.) is reached
647.            }
648.      //converter a soma do MSB(1byte)+LSB(1byte) em inteiro (4Bytes)
649.      unsigned char byte;
650.            byte =i2c_inbyte(ACK);        //read the first byte (MSB)
651.            *p_value=byte;
652.            *p_value *=256;
653.            byte=i2c_inbyte(ACK);         //read the second byte (LSB)
654.            *p_value +=byte;
655.            *p_checksum =i2c_inbyte(noACK);           //read checksum
656.      return errorH;
657.      }
658.      // calculates temperature [ C] and humidity [%RH]
659.      // input : humi [Ticks] (12 bit)
660.      //         temp [Ticks] (14 bit)
661.      // output: humi [%RH]
662.      //         temp [ C]
663.      void calc_sth11(float *p_humidity ,float *p_temperature)
664.      {
665.      const float C1=-4.0;          // for 12 Bit
666.      const float C2= 0.0405;       // for 12 Bit
667.      const float C3=-0.0000028;    // for 12 Bit
668.      const float T1=0.01;          // for 14 Bit @ 5V
669.      const float T2=0.00008;       // for 14 Bit @ 5V
670.      float  rh=*p_humidity;        // rh:      Humidity [Ticks] 12
Bit
671.      float  t=*p_temperature;      // t:       Temperature [Ticks] 14
Bit
672.      float  rh_lin;                // rh_lin:  Humidity linear
673.      float  rh_true;               // rh_true: Temperature
compensated humidity
674.      float  t_C;             // t_C   :  Temperature [ C]
675.            t_C=t*0.01-40.0;                          //calc.
Temperature from ticks to [ C]
676.            rh_lin=C3*rh*rh+C2*rh+C1;                 //calc. Humidity
from ticks to [%RH]
```

```
677.            rh_true=(t_C-25)*(T1+T2*rh)+rh_lin; //calc. Temperature
compensated humidity [%RH]
678.            if(rh_true>100)
679.                  rh_true=100;                              //cut if
the value is outside of
680.            if(rh_true<0.1)
681.                  rh_true=0.1;                              //the
physical possible range
682.            *p_temperature=t_C;                       //return
temperature [ C]
683.            *p_humidity=rh_true;                      //return
humidity[%RH]
684.      }
685.      // calculates dew point
686.      // input:    humidity [%RH], temperature [ C]
687.      // output: dew point [ C]
688.      float calc_dewpoint(float h,float t)
689.      {
690.      float k;
691.            k = (log10(h)-2)/0.4343 + (17.62*t)/(243.12+t);
692.            dew_point = 243.12*k/(17.62-k);
693.            return dew_point;
694.      }
695.      //Rotina para escrever na md03
696.      void md03_i2c(unsigned char endereco, unsigned char registo,
unsigned char valor)
697.      {
698.            i2c_start();
699.            if (i2c_outbyte(endereco)==0)
700.            {
701.                  printf("Escrita MD03: NACK received %d\r",__LINE__);
702.                  i2c_stop();
703.            }
704.            if (i2c_outbyte(registo)==0)
705.            {
706.                  printf("Escrita MD03: NACK received %d\r",__LINE__);
707.                  i2c_stop();
708.            }
709.            if (i2c_outbyte(valor)==0)
710.            {
711.                  printf("Escrita MD03: NACK received %d\r",__LINE__);
```

```
712.                  i2c_stop();
713.            }
714.            i2c_stop();
715.      }
716.      //Rotina para ler a md03
717.      int LerMD03(unsigned char endereco, unsigned char endereco1,
unsigned char registo)
718.      {
719.            unsigned int a;
720.            i2c_start();                  //mandar sinal de inicio
721.            if (i2c_outbyte(endereco)==0)
722.            {                             //Escrever qual Ã© o
endereÃ§o da MD03
723.                  printf("Leitura MD03: NACK received %d\r",__LINE__);
724.                  i2c_stop();
725.            }
726.            if (i2c_outbyte(registo)==0)
727.            {                       //Escrever o numero do registo que
se pretende ler
728.                  printf("Leitura MD03: NACK received %d\r",__LINE__);
729.                  i2c_stop();
730.            }
731.            i2c_start();                  //Repetir o sinal de inicio
732.            if (i2c_outbyte(endereco1)==0)
733.            {                       //Escrever qual Ã© o endereÃ§o da
MD03 no modo leitura 0xC1
734.                  printf("Leitura MD03: NACK received %d\r",__LINE__);
735.                  i2c_stop();
736.            }
737.            a=i2c_inbyte(1);                          //Ler informacao
738.      i2c_stop();
739.      return a;
740.      }
741.      int abrir_i2c(void)
742.      {
743.            if (i2c_open()<0)
744.            {
745.                  printf("i2c open error\r");
746.                  return 1;
747.            }
748.            else
```

```
749.                  return 0;
750.      }
751.      //Ler Valores de I e Temp das MD03
752.      void LerTem_Current()
753.      {
754.            ID= LerMD03(MOTOR_DIR,0xB3,0x05);
755.            ID=((ID*20)/186);
756.            usleep(100);
757.            IE= LerMD03(MOTOR_ESQ,0xB1,0x05);
758.            IE=((IE*20)/186);
759.            usleep(100);
760.            TEMPD= LerMD03(MOTOR_DIR,0xB3,0x04);
761.            te=TEMPD;
762.            usleep(100);
763.            TEMPE= LerMD03(MOTOR_ESQ,0xB1,0x04);
764.            td=TEMPE;
765.      }
766.      //Fazer andar o Robo
767.      void anda(int veloD, int veloE )
768.      {
769.      printf("VeloD = %4d VeloE=%4d\r", veloD, veloE);
770.            md03_i2c(MOTOR_ESQ,0x02,(abs(veloE)));
771.            md03_i2c(MOTOR_ESQ,0x00,(veloE>0?1:2));
772.            md03_i2c(MOTOR_DIR,0x02,(abs(veloD)));
773.            md03_i2c(MOTOR_DIR,0x00,(veloD>0?1:2));
774.      }
775.      //Ler InformaÃ§Ã£o da Bussola
776.      void LerBusola(void)
777.      {
778.            i2c_start();                  //mandar sinal de inicio
779.            if (i2c_outbyte(0xC0)==0) {
   //Escrever qual Ã© o endereÃ§o da Busola 0xC0
780.                  printf("Busola: NACK received %d\r",__LINE__);
781.                  i2c_stop();
782.            }
783.            if (i2c_outbyte(0x01)==0)
784.            {                       //Escrever o numero do registo que
se pretende ler
785.                  printf("Bussola: NACK received %d\r",__LINE__);
786.                  i2c_stop();
787.            }
```

```
788.            i2c_start();                                        //Repetir
o sinal de inicio
789.            if (i2c_outbyte(0xC1)==0)
790.            {                       //Escrever qual Ã© o endereÃ§o da
Busola no modo leitura 0xC1
791.                  printf("Bussola: NACK received %d\r",__LINE__);
792.                  i2c_stop();
793.            }
794.            Bussola = i2c_inbyte(1);
795.            i2c_stop();
796.      }
797.      //Ler Valor dos ultra-sons
798.      void Range_Sonar(unsigned char enderecoW,unsigned char
enderecoR,unsigned char registoW,unsigned char registoR, unsigned char
valor)
799.      {
800.      sonar = 0;
801.      i2c_start();
                  //Start
802.            if (i2c_outbyte(enderecoW)==0)
803.            {
804.                  printf("NACK received %d\r",__LINE__);
805.                  i2c_stop();
806.            }
807.            if (i2c_outbyte(registoW)==0)
808.            {
809.                  printf("\nNACK received %d\r",__LINE__);
810.                  i2c_stop();
811.            }
812.            if (i2c_outbyte(valor)==0)
813.            {
814.                  printf("NACK received %d\r",__LINE__);
815.                  i2c_stop();
816.            }
817.      usleep(10000);//Esperar 10ms para que o ranging acab
818.            i2c_stop();
819.            i2c_start();
820.            if (i2c_outbyte(enderecoW)==0)
821.            {                 //Mandar o EndereÃ§o do dispositivo
822.                  printf("NACK received %d\r",__LINE__);
823.                  i2c_stop();
```

```
824.           }
825.           if (i2c_outbyte(registoR)==0)
826.           {                       //Mandar o Registo sobre o qual se
efectua a leitura
827.                 printf("NACK received %d\r",__LINE__);
828.                 i2c_stop();
829.           }
830.           i2c_start();
   //Repetir o Start
831.           if (i2c_outbyte(enderecoR)==0)
832.           {                 //Mandar o EndereÃ§o do dispositivo com
modo de leitura activo
833.                 printf("NACK received %d\r",__LINE__);
834.                 i2c_stop();
835.           }
836.           sonar=i2c_inbyte(1);
837.           i2c_stop();
838.     }
839.     //INICIALIZA JOYSTIC
840.     void startjoystic()
841.     {
842.     int iomask;
843.           iomask=1<<5;            //Activa XBEE
844.           printf("RESET ON\r");
845.           ioctl(i2c_fd,_IO(ETRAXGPIO_IOCTYPE,IO_SETBITS),iomask);
846.           printf("baud=%d\r", getbaud(fd));
847.           initport(fd);
848.           printf("baud=%d\r", getbaud(fd));
849.                 if (!init_comandoAT(fd))
850.                 {
851.                       printf("Inicio '+++' Comando AT falhou\r");
852.                 }
853.           char sCmd[254]={"ATID3330"};
854.                 if (!comandoAT(fd, sCmd))
855.                 {
856.                       printf("Comando ATDL falhou\r");
857.                 }
858.           char sCmd1[254]={"ATMY1"};
859.                 if (!comandoAT(fd, sCmd1))
860.                 {
861.                       printf("Comando ATCN falhou\r");
```

```
862.                 }
863.           char sCmd3[254]={"ATCN"};
864.                 if (!comandoAT(fd, sCmd3))
865.                 {
866.                       printf("Comando ATCN falhou\r");
867.                 }
868.     }
869.     //Sensor de Humidade
870.     void Calcula_Humidade(void)
871.     {
872.     unsigned char errorH,checksum;
873.     int humi_i, temp_i;
874.     humi_i=0; humi=0.0;
875.     temp_i=0; temp=0.0;
876.     errorH=0;
877.                 errorH+=s_measure(&humi_i,&checksum,HUMI); //measure
humidity
878.                 errorH+=s_measure(&temp_i,&checksum,TEMP); //measure
temperature
879.                 if(errorH!=0) // Se erro>=1 Ã© sinal que houve erro
em algum lado...
880.                 {
881.                       printf("\ndeuERRO");
882.                       s_connectionreset();                  //in
case of an error: connection reset
883.                 }
884.                 else
885.                 {
886.                       humi=(float)humi_i;
   //converts integer to float
887.                       temp=(float)temp_i;
   //converts integer to float
888.
889.                       calc_sth11(&humi,&temp);
   //calculate humidity, temperature
890.                       dew_point=calc_dewpoint(humi,temp);
   //calculate dew point
891.                 }
892.     }
893.     //THREAD JOYSTIC
894.     void thread_joystic(void)
```

```
895.      {
896.      int x=0;
897.      int y=0;
898.      int z=0;
899.      int vel1=0,dir1=0;
900.      char sResult1[50];
901.      int count=0;
902.            while(a_correr)
903.            {
904.            if(Flag_stop_joystic == 0)
905.            {
906.                  usleep(200);
907.                  fcntl(fd, F_SETFL, FNDELAY); // don't block serial
read
908.                  if (readport(fd,sResult1))
909.                  {
910.                        count=0;
911.                        sscanf(sResult1,"%*s %d %*s %d %*s %d", &z,
&y, &x);
912.                        strcpy(sResult1," ");
913.                        vel1=y*-2;
914.                        dir1=x*-0.6;
915.                        if(vel1 < 20 && vel1 > -20)
916.                        {
917.                              vel = 0; dir = 0;
918.                              vel = 0;
919.                              velD = 0;
920.                              velE = 0;
921.                              velocidadeD=0; velocidadeE=0;
922.                              velocidade_E = 0, velocidade_D = 0;
923.                        }
924.                        else
925.                        {
926.                              vel = vel1;
927.                        }
928.                        if(dir1 < 10 && dir1 > -10)
929.                        {
930.                              dir = 0;
931.                        }
932.                        else
933.                        {
```

```
                              dir = dir1;
                        }
                        limita_vel(dir,vel);
      printf("\t\t\tvel = ____%d_______ dir =
_____%d_________\r",vel,dir);
                  }
                  else
                  {
                        count++;

                        if(count > 15)
                        {
                              printf("Falha na Comunicacao\r");

                              vel = 0; dir = 0;
                              vel = 0;
                              velD = 0;
                              velE = 0;
                              velocidadeD=0; velocidadeE=0;
                              velocidade_E = 0, velocidade_D = 0;
                              limita_vel(dir,vel);
                        }
                        else
                        {
                              printf("Sem dados\r");
                        }
                  }
            }
            }//Fim ciclo While
      }
      //Recebe ordens do PC com o processamento de imagem quando a
comunicaÃ§Ã£o estiver ligada
      void thread_comunica(void)
      {
      int sock, length, fromlen, n;
      struct sockaddr_in server;
      struct sockaddr_in from;
      char buf[1024];
            sock=socket(AF_INET, SOCK_DGRAM, 0);
            if (sock < 0)
                  error("Opening socket");
```

```
973.            length = sizeof(server);
974.            bzero(&server,length);
975.            server.sin_family=AF_INET;
976.            server.sin_addr.s_addr=INADDR_ANY;
977.            server.sin_port=htons(50000);
978.            if (bind(sock,(struct sockaddr *)&server,length)<0)
979.                  error("binding");
980.            fromlen = sizeof(struct sockaddr_in);
981.            while (a_correr)
982.            {
983.                  if(Flag_cala_t == 0 && Flag_comunica1 == 0)
984.                  {
985.                        n = recvfrom(sock,buf,1024,0,(struct sockaddr
*)&from,&fromlen);
986.                        sscanf(buf,"%*s %d %*s %d", &vel, &dir);
987.      printf("\nRecebido= %s",buf);
988.                        limita_vel(dir,vel);
989.                        if (n < 0)
990.                        error("recvfrom");
991.                  }
992.            }
993.            close(sock);
994.      }
995.      /*****************************************************************
*******************************************/
996.      //Verifica qual a inclinaÃ§Ã£o do RobÃ´
997.      //Pan e Tilt respectivamente
998.      void thread_AcelY(void)
999.      {
1000.           printf("\nInicio da thread Acelarometro\n");
1001.
1002.     time_t T1Y=0,T2Y=0;
1003.     float DutyY_=0.00,DutyY=0.00;
1004.     int valueY=0, value_inicialY=0;
1005.     time_t curtimeY=0,curtimefY=0,curtimef2Y=0;
1006.
1007.                 //      T1X T2X   T3
1008.                 //    _____
1009.                 //    |        |       |
1010.                 //____|        |_______|
1011.                 //
```

```
1012.                 //Ler o Tempo em 3 instantes de TransiÃ§Ã£o
1013.                 //Fazer sustracÃ§Ã£o e obter os Tempos
1014.           while(a_correr)
1015.           {
1016.                 value_inicialY=!!(ioctl(fdPin,
_IO(ETRAXGPIO_IOCTYPE, IO_READBITS)) & iomaskY);
1017.                 valueY=value_inicialY;
1018.                 while(value_inicialY==valueY)
1019.                 {
1020.                       valueY=!!(ioctl(fdPin, _IO(ETRAXGPIO_IOCTYPE,
IO_READBITS)) & iomaskY);
1021.                 }
1022.                 gettimeofday(&tv, NULL);
1023.                 curtimeY=tv.tv_usec;
1024.                 value_inicialY=valueY;
1025.                 while(value_inicialY==valueY)
1026.                 {
1027.                       valueY=!!(ioctl(fdPin, _IO(ETRAXGPIO_IOCTYPE,
IO_READBITS)) & iomaskY);
1028.                 }
1029.                 gettimeofday(&tv, NULL);
1030.                 curtimefY=tv.tv_usec;
1031.                 value_inicialY=valueY;
1032.                 while(value_inicialY==valueY)
1033.                 {
1034.                       valueY=!!(ioctl(fdPin, _IO(ETRAXGPIO_IOCTYPE,
IO_READBITS)) & iomaskY);
1035.                 }
1036.                 gettimeofday(&tv, NULL);
1037.                 curtimef2Y=tv.tv_usec;
1038.                 //DiferenÃ§a entre os tempos  T1X
1039.                 if(curtimefY<curtimeY)
1040.                 {
1041.                       T1Y=(1000000-curtimeY)+curtimefY;
1042.                 }
1043.                 else
1044.                 {
1045.                       T1Y=curtimefY-curtimeY;
1046.                 }
1047.                 if(curtimef2Y<curtimefY)
1048.                 {
```

```
1049.                       T2Y=(1000000-curtimeY)+curtimefY;
1050.                 }
1051.                 else
1052.                 {
1053.                       T2Y=curtimef2Y-curtimefY;
1054.                 }
1055.                 if(valueY==0)
1056.                 {
1057.                       //Tempo passado a alto T2X
1058.                       //Tempo passado a baixo T1X
1059.                       DutyY_=((float)T2Y/((float)T2Y+(float)T1Y));
1060.                 }
1061.                 else
1062.                 {
1063.                             //Tempo passado a baixo T1X
1064.                             //Tempo passado a alto T2X
1065.                       DutyY_=((float)T1Y/((float)T2Y+(float)T1Y));
1066.                 }
1067.                 aceY++;
1068.                 vy[aceY]=DutyY_;
1069.                 if (aceY==5)
1070.                 {
1071.                       float ay=0,by=1,my=0;
1072.                       int iy;
1073.                             for(iy=1;iy<=5;iy++)
1074.                             {
1075.                                   if(ay<=vy[iy])
1076.                                   {ay=vy[iy];}
1077.                                   if(by>=vy[iy])
1078.                                   {by=vy[iy];}
1079.                                   my=my+vy[iy];
1080.                             }
1081.                             my=(my-ay-by)/3;
1082.                             if((ay<=by+0.05)&&(by>=ay-0.05)) //Se a
diferenÃ§a entre o valor maximo e minimo for grande ignora leitura
1083.                             {
1084.                                   DutyY=my;
1085.                                   gY=(((1-DutyY)-0.5)/0.125);
1086.                                   Pan=asin((double)gY);
1087.                                   Pan=(((180)*Pan)/3.14);
1088.                             }
```

```
1089.                           aceY=0;
1090.                }
1091.          usleep(100000);
1092.          }
1093.     }
1094.     void thread_AcelX(void)
1095.     {
1096.          printf("\nInicio da thread Acelarometro\n");
1097.
1098.     time_t T1X=0,T2X=0;
1099.     float DutyX_=0.0,DutyX=0.0;
1100.     int valueX=0, value_inicialX=0;
1101.     time_t curtimeX=0,curtimefX=0,curtimef2X=0;
1102.
1103.                //      T1X T2X   T3
1104.                //     _____
1105.                //    |           |     |
1106.                //____|           |_______|
1107.                //
1108.                //Ler o Tempo em 3 instantes de TransiÃ§Ã£o
1109.                //Fazer sustracÃ§Ã£o e obter os Tempos
1110.          while(a_correr)
1111.                {
1112.                     value_inicialX=!!(ioctl(fdPin,
_IO(ETRAXGPIO_IOCTYPE, IO_READBITS)) & iomaskX);
1113.                     valueX=value_inicialX;
1114.                     while(value_inicialX==valueX)
1115.                     {
1116.                          valueX=!!(ioctl(fdPin,
_IO(ETRAXGPIO_IOCTYPE, IO_READBITS)) & iomaskX);
1117.                     }
1118.                     gettimeofday(&tv, NULL);
1119.                     curtimeX=tv.tv_usec;
1120.                     value_inicialX=valueX;
1121.                     while(value_inicialX==valueX)
1122.                     {
1123.                          valueX=!!(ioctl(fdPin,
_IO(ETRAXGPIO_IOCTYPE, IO_READBITS)) & iomaskX);
1124.                     }
1125.                     gettimeofday(&tv, NULL);
1126.                     curtimefX=tv.tv_usec;
```

```
1127.                       value_inicialX=valueX;
1128.                       while(value_inicialX==valueX)
1129.                       {
1130.                             valueX=!!(ioctl(fdPin,
_IO(ETRAXGPIO_IOCTYPE, IO_READBITS)) & iomaskX);
1131.                       }
1132.                       gettimeofday(&tv, NULL);
1133.                       curtimef2X=tv.tv_usec;
1134.                       //DiferenÃ§a entre os tempos  T1X
1135.                       if(curtimefX<curtimeX)
1136.                       {
1137.                             T1X=(1000000-curtimeX)+curtimefX;
1138.                       }
1139.                       else
1140.                       {
1141.                             T1X=curtimefX-curtimeX;
1142.                       }
1143.                       if(curtimef2X<curtimefX)
1144.                       {
1145.                             T2X=(1000000-curtimeX)+curtimefX;
1146.                       }
1147.                       else
1148.                       {
1149.                             T2X=curtimef2X-curtimefX;
1150.                       }
1151.                       if(valueX==0)
1152.                       {
1153.                                   //Tempo passado a alto T2X
1154.                                   //Tempo passado a baixo T1X
1155.
   DutyX_=((float)T2X/((float)T2X+(float)T1X));
1156.                       }
1157.                       else
1158.                       {
1159.                                   //Tempo passado a baixo T1X
1160.                                   //Tempo passado a alto T2X
1161.
   DutyX_=((float)T1X/((float)T2X+(float)T1X));
1162.                       }
1163.                 aceX++;
1164.                 vx[aceX]=DutyX_;
```

```
1165.                       if (aceX==5)
1166.                       {
1167.                             float ax=0,bx=1,mx=0;
1168.                             int ix;
1169.                                   for(ix=1;ix<=5;ix++)
1170.                                   {
1171.                                         if(ax<=vx[ix])
1172.                                         {ax=vx[ix];}
1173.                                         if(bx>=vx[ix])
1174.                                         {bx=vx[ix];}
1175.                                         mx=mx+vx[ix];
1176.                                   }
1177.                             mx=(mx-ax-bx)/3;
1178.                             if((ax<=bx+0.05)&&(bx>=ax-0.05))
1179.                             {
1180.                                   DutyX=mx;
1181.                                   gX=(((1-DutyX)-0.5)/0.125);
1182.                                   Tilt=asin((double)gX);
1183.                                   Tilt=(((180)*Tilt)/3.14);
1184.                             }
1185.                             aceX=0;
1186.                 }
1187.           usleep(100000);
1188.           }
1189.     }
1190.     //ENCODERS
1191.     float velocidade_EM = 0.00;
1192.     void thread_encoderE(void)
1193.     {
1194.     float erroE=0.00, erroE_anterior=0.00;
1195.     float velocidadeE0=0.00, velocidade_auxE= 0.00;
1196.     float veloE[2],erroCE = 0;
1197.     printf("\nInicio da thread Encoder D\n");
1198.     int timeoutE=0;
1199.     //float erroE=0.00, erroE_anterior=0.00;
1200.     int valueE=0, value_inicialE=0;
1201.     int iE = 0;
1202.     suseconds_t curtimeE=0,
curtimefE=0,tempo_1voltaE=0,tempo_voltaE=0;
1203.     printf("\n\n\t\t\tKPE=%.2f KIE=%.2f\n\n",KPE,KIE);
1204.           while(a_correr)
```

```
1205.           {
1206.     //Calculo da velocidade recorrendo ao encoder
1207.                 timeoutE=0;
1208.                 if(velocidade_E == 0.00)
1209.                 {
1210.                       velocidadeE = velE;
1211.                 }
1212.                 valueE = !!(ioctl(fdPin, _IO(ETRAXGPIO_IOCTYPE,
IO_READBITS)) & iomaskE);
1213.                 value_inicialE = valueE;
1214.                 while((value_inicialE == valueE) && (timeoutE <
1000))
1215.                 {
1216.                       valueE = !!(ioctl(fdPin,
_IO(ETRAXGPIO_IOCTYPE, IO_READBITS)) & iomaskE); //para obter o tempo
sempre no mesmo ponto
1217.                       timeoutE ++;
1218.                 }
1219.                 if(timeoutE < 950)
1220.                 {
1221.                       if(gettimeofday(&tv, NULL) != 0)
1222.                       {
1223.                             printf("\n\t\t\t\tTIME_ERRO");
1224.                       }
1225.                       curtimeE = tv.tv_usec;
1226.                       timeoutE = 0;
1227.                       valueE = !!(ioctl(fdPin,
_IO(ETRAXGPIO_IOCTYPE, IO_READBITS)) & iomaskE);
1228.                       value_inicialE = valueE;
1229.                       while((value_inicialE == valueE) && (timeoutE
< 1000))
1230.                       {
1231.                             valueE = !!(ioctl(fdPin,
_IO(ETRAXGPIO_IOCTYPE, IO_READBITS)) & iomaskE);
1232.                             timeoutE ++;
1233.                       }
1234.                       timeoutE = 0;
1235.                       valueE = !!(ioctl(fdPin,
_IO(ETRAXGPIO_IOCTYPE, IO_READBITS)) & iomaskE);
1236.                       value_inicialE = valueE;
```

```
1237.                       while((value_inicialE == valueE) && (timeoutE
< 1000) )
1238.                       {
1239.                             valueE = !!(ioctl(fdPin,
_IO(ETRAXGPIO_IOCTYPE, IO_READBITS)) & iomaskE);
1240.                             timeoutE ++;
1241.                       }
1242.                       if(gettimeofday(&tv, NULL) != 0)
1243.                       {
1244.                             printf("\n\tTIME_ERRO");
1245.                       }
1246.                       curtimefE=tv.tv_usec;
1247.                       if(curtimefE < curtimeE)
1248.                       {
1249.                             tempo_voltaE = (1000000 + curtimefE) -
curtimeE;
1250.                       }
1251.                       else
1252.                       {
1253.                             tempo_voltaE = curtimefE - curtimeE;
1254.                       }
1255.                       tempo_1voltaE = (512) * tempo_voltaE;
1256.                       velocidade_E = ((2*M_PI*0.2) /
(((float)tempo_1voltaE)/1000000));      //2*3.1415*0.2    velocidade
actual
1257.                       velocidade_E = (velocidade_E * 100) + 30; //30
1258.                 }
1259.                 else
1260.                 {
1261.                       velocidade_E = 0;
1262.                 }
1263.                 velocidade_EE = velocidade_E;
1264.                             velocidade_EM = velocidade_E;
//((veloE[0] + veloE[1])/2);
1265.                             erroE = velE - velocidade_EM;
1266.                             if((erroE < 5) && (erroE > -5))
1267.                             {
1268.                                   erroE = 0;
1269.                                   erroE_anterior = 0;
1270.                                   velocidadeE0 = velE;
1271.                             }
```

```
1272.                           if(vel > 80 && flagcontrolo == 1)
1273.                           {
1274.                                 velocidade_auxE = velocidadeE0 +
(float)KPE * erroE + (float)KIE * erroE_anterior;
1275.                           }
1276.                           else
1277.                           {
1278.                                 velocidade_auxE = velE; //para
retirar o controlo
1279.                           }
1280.                           erroE_anterior = erroE;
1281.                           if(velocidade_auxE > 250.0)
1282.                           {
1283.                                 velocidadeE = 250;
1284.                           }
1285.                           if(velocidade_auxE < -250.0)
1286.                           {
1287.                                 velocidadeE = -250;
1288.                           }
1289.                           if(velocidade_auxE < 250.0 &&
velocidade_auxE > -250.0)
1290.                           {
1291.                                 velocidadeE = velocidade_auxE;
1292.                           }
1293.                           velocidadeE0 = velocidadeE;
1294.           }//Fim do while
1295.     }//Fim da MainE
1296.     float velocidade_DM = 0.00;
1297.     void thread_encoderD(void)
1298.     {
1299.     float erroD=0.00, erroD_anterior=0.00;
1300.     float velocidadeD0=0.00, velocidade_auxD= 0.00;
1301.     float veloD[2],erroCD = 0;
1302.     printf("\nInicio da thread Encoder D\n");
1303.     int timeoutD=0;
1304.     //float erroD=0.00, erroD_anterior=0.00;
1305.     int valueD=0, value_inicialD=0;
1306.     int iD = 0;
1307.     suseconds_t curtimeD=0,
curtimefD=0,tempo_1voltaD=0,tempo_voltaD=0;
1308.     printf("\n\n\t\t\tKPD=%.2f KID=%.2f\n\n",KPD,KID);
```

```
1309.           while(a_correr)
1310.           {
1311.     //Calculo da velocidade recorrendo ao encoder
1312.                 timeoutD=0;
1313.                 if(velocidade_D == 0.00)
1314.                 {
1315.                       velocidadeD = velD;
1316.                 }
1317.                 valueD = !!(ioctl(fdPin, _IO(ETRAXGPIO_IOCTYPE,
IO_READBITS)) & iomaskD);
1318.                 value_inicialD = valueD;
1319.                 while((value_inicialD == valueD) && (timeoutD <
1000))
1320.                 {
1321.                       valueD = !!(ioctl(fdPin,
_IO(ETRAXGPIO_IOCTYPE, IO_READBITS)) & iomaskD); //para obter o tempo
sempre no mesmo ponto
1322.                       timeoutD ++;
1323.                 }
1324.                 if(timeoutD < 950)
1325.                 {
1326.                       if(gettimeofday(&tv, NULL) != 0)
1327.                       {
1328.                             printf("\n\t\t\t\tTIME_ERRO");
1329.                       }
1330.                       curtimeD = tv.tv_usec;
1331.                       timeoutD = 0;
1332.                       valueD = !!(ioctl(fdPin,
_IO(ETRAXGPIO_IOCTYPE, IO_READBITS)) & iomaskD);
1333.                       value_inicialD = valueD;
1334.                       while((value_inicialD == valueD) && (timeoutD
< 1000))
1335.                       {
1336.                             valueD = !!(ioctl(fdPin,
_IO(ETRAXGPIO_IOCTYPE, IO_READBITS)) & iomaskD);
1337.                             timeoutD ++;
1338.                       }
1339.                       timeoutD = 0;
1340.                       valueD = !!(ioctl(fdPin,
_IO(ETRAXGPIO_IOCTYPE, IO_READBITS)) & iomaskD);
1341.                       value_inicialD = valueD;
```

```
1342.                       while((value_inicialD == valueD) && (timeoutD
< 1000) )
1343.                       {
1344.                             valueD = !!(ioctl(fdPin,
_IO(ETRAXGPIO_IOCTYPE, IO_READBITS)) & iomaskD);
1345.                             timeoutD ++;
1346.                       }
1347.                       if(gettimeofday(&tv, NULL) != 0)
1348.                       {
1349.                             printf("\n\tTIME_ERRO");
1350.                       }
1351.                       curtimefD=tv.tv_usec;
1352.                       if(curtimefD < curtimeD)
1353.                       {
1354.                             tempo_voltaD = (1000000 + curtimefD) -
curtimeD;
1355.                       }
1356.                       else
1357.                       {
1358.                             tempo_voltaD = curtimefD - curtimeD;
1359.                       }
1360.                       tempo_1voltaD = (512) * tempo_voltaD;
1361.                       velocidade_D = ((2*M_PI*0.2) /
(((float)tempo_1voltaD)/1000000));      //2*3.1415*0.2    velocidade
actual
1362.                       velocidade_D = (velocidade_D * 100) + 30;
//30
1363.                 }
1364.                 else
1365.                 {
1366.                       velocidade_D = 0;
1367.                 }
1368.                 velocidade_ED = velocidade_D;
1369.                             velocidade_DM =velocidade_D ;
//((veloD[0] + veloD[1])/2);
1370.                             //Controlo PI
1371.                             erroD = velD - velocidade_DM;
1372.                             if((erroD < 5) && (erroD > -5))
1373.                             {
1374.                                   erroD = 0;
1375.                                   erroD_anterior = 0;
```

```
1376.                                   velocidadeD0 = velD;
1377.                             }
1378.                             if(vel > 80 && flagcontrolo == 1)
1379.                             {
1380.                                   velocidade_auxD = velocidadeD0 +
(float)KPD * erroD + (float)KID * erroD_anterior;
1381.                             }
1382.                             else
1383.                             {
1384.                                   velocidade_auxD = velD; //para
retirar o controlo
1385.                             }
1386.                             erroD_anterior = erroD;
1387.                             if(velocidade_auxD > 250.0)
1388.                             {
1389.                                   velocidadeD = 250;
1390.                             }
1391.                             if(velocidade_auxD < -250.0)
1392.                             {
1393.                                   velocidadeD = -250;
1394.                             }
1395.                             if(velocidade_auxD < 250.0 &&
velocidade_auxD > -250.0)
1396.                             {
1397.                                   velocidadeD = velocidade_auxD;
1398.                             }
1399.                             velocidadeD0 = velocidadeD;
1400.           }//Fim do while
1401.     }//Fim da Main
1402.     //Verifica qual o estado dos fusÃ-veis
1403.     void thread_Actualiza_Fusiveis(void)
1404.     {
1405.     int FusivelM1 = 0, FusivelM2 = 0, FusivelM3 = 0, EstadoPotencia
= 0;
1406.     int iomaskF1 = 1<<22,iomaskF2 = 1<<20,iomaskF3 = 1<<18, iomaskP
= 1<<16;
1407.     //IOG22; IOG20; IOG18; IOG16; IOG17
1408.     //4 fusÃ-veis e um pino para activar ou desactivar a carga das
baterias...
1409.     char str[1024];
1410.     suseconds_t time0=0, time1=0;
```

```
1411.          while(a_correr)
1412.                {
1413.                      vel_linear = (velocidade_ED + velocidade_EE) /
2; //VELOCIDADE LINEAR DO ROBO”
1414.
1415.                      FusivelM1 = !!(ioctl(fdPin,
_IO(ETRAXGPIO_IOCTYPE, IO_READBITS)) & iomaskF1);
1416.                      FusivelM2 = !!(ioctl(fdPin,
_IO(ETRAXGPIO_IOCTYPE, IO_READBITS)) & iomaskF2);
1417.                      FusivelM3 = !!(ioctl(fdPin,
_IO(ETRAXGPIO_IOCTYPE, IO_READBITS)) & iomaskF3);
1418.                      EstadoPotencia = !!(ioctl(fdPin,
_IO(ETRAXGPIO_IOCTYPE, IO_READBITS)) & iomaskP);
1419.                }
1420.    }
1421.    //InicializaÃ§Ã£o USB0 para o GPS
1422.    void CloseAdrPort()
1423.    {
1424.          // you may want to restore the saved port attributes
1425.       if (fdGPS > 0)
1426.        {
1427.            close(fdGPS);
1428.        } // end if
1429.    } // end CloseAdrPort
1430.    int OpenAdrPort(char* sPortNumber)
1431.    {
1432.        char sPortName[64];
1433.        printf("in OpenAdrPort port#=%s\n", sPortNumber);
1434.        sprintf(sPortName, "/dev/ttyUSB%s", sPortNumber);
1435.        printf("sPortName=%s\n", sPortName);
1436.        CloseAdrPort(fdGPS);
1437.        fdGPS = open(sPortName, O_RDWR | O_NOCTTY | O_NDELAY);
1438.        if (fdGPS < 0)
1439.        {
1440.            printf("open error %d %s\n", errno, strerror(errno));
1441.        }
1442.        else
1443.        {
1444.          struct termios my_termios;
1445.                // Get the current options for the port...
1446.                tcgetattr(fdGPS, &my_termios);
```

```
1447.                // Set the baud rates to 4800...
1448.                cfsetispeed(&my_termios, B4800); //
1449.                cfsetospeed(&my_termios, B4800); //
1450.                // Enable the receiver and set local mode...
1451.                my_termios.c_cflag |= (CLOCAL | CREAD);
1452.                my_termios.c_cflag &= ~PARENB;
1453.                my_termios.c_cflag &= ~CSTOPB;
1454.                my_termios.c_cflag &= ~CSIZE;
1455.                my_termios.c_cflag |= CS8;
1456.                // Set the new options for the port...
1457.                tcsetattr(fdGPS, TCSANOW, &my_termios);
1458.        } // end if
1459.        return fdGPS;
1460.    } // end OpenAdrPort
1461.    int WriteAdrPort(char* psOutput)
1462.    {
1463.        int iOut;
1464.        if (fdGPS < 1)
1465.        {
1466.            printf(" port is not open\n");
1467.            return -1;
1468.        } // end if
1469.        iOut = write(fdGPS, psOutput, strlen(psOutput));
1470.        if (iOut < 0)
1471.        {
1472.            printf("write error %d %s\n", errno, strerror(errno));
1473.        }
1474.        else
1475.        {
1476.          //printf("wrote %d chars: %s\n", iOut, psOutput);
1477.        } // end if
1478.        return iOut;
1479.    } // end WriteAdrPort
1480.    int ReadAdrPort(char* psResponse, int iMax)
1481.    {
1482.    int iIn;
1483.    //    printf("iMax=%d\t", iMax);
1484.          if (fdGPS < 1)
1485.          {
1486.            printf("GPS port is not open__ \r");
1487.            return -1;
```

```
1488.           } // end if
1489.           strncpy (psResponse, "N/A", iMax<4?iMax:4);
1490.           iIn = read(fdGPS, psResponse, iMax-1);
1491.           if (iIn < 0)
1492.           {
1493.                 if (errno == EAGAIN)
1494.                 {
1495.                       return 0; // assume that command generated no
response
1496.                 }
1497.                 else
1498.                 {
1499.                       printf("GPS read error %d %s__\r", errno,
strerror(errno));
1500.                 } // end if
1501.           }
1502.           else
1503.           {
1504.                 psResponse[iIn<iMax?iIn:iMax] = '\0';
1505.             printf("read %d chars: %s\n", iIn, psResponse);
1506.           } // end if
1507.     return iIn;
1508.     } // end ReadAdrPort
1509.     TRAMA_GPS converte_Km(char* strGPS)
1510.     {
1511.     double lati, longi;
1512.     char aux[5];
1513.     int i = 0;
1514.           for(i=0; i<5; i++)      //limpa a string
1515.                 aux[i] = ' ';
1516.           aux[0] = strGPS[16];
1517.           aux[1] = strGPS[17];
1518.           gps.latitude = atof(aux);     //latitude em graus
1519.           aux[0] = strGPS[18];
1520.           aux[1] = strGPS[19];
1521.           gps.latitude += atof(aux)/60;  //converte minutos para
graus
1522.           aux[0] = strGPS[21];
1523.           aux[1] = strGPS[22];
1524.           aux[2] = '.';
1525.           aux[3] = strGPS[23];
```

```
1526.           aux[4] = strGPS[24];
1527.           gps.latitude += atof(aux)/3600;  //converte segundos para
graus
1528.           for(i=0; i<5; i++)      ////limpa a string
1529.           aux[i] = ' ';
1530.           aux[0] = strGPS[28];    //Faz o mesmo para a longitude
1531.           aux[1] = strGPS[29];
1532.           aux[2] = strGPS[30];
1533.           gps.longitude = atof(aux);
1534.           aux[0] = strGPS[31];
1535.           aux[1] = strGPS[32];
1536.           aux[2] = ' ';
1537.           gps.longitude += atof(aux)/60;
1538.           aux[0] = strGPS[34];
1539.           aux[1] = strGPS[35];
1540.           aux[2] = '.';
1541.           aux[3] = strGPS[36];
1542.           aux[4] = strGPS[37];
1543.           gps.longitude += atof(aux)/3600;
1544.           lati = 6378*sin(gps.latitude*M_PI/180);
1545.           longi =
6378*cos(gps.latitude*M_PI/180)*sin(gps.longitude*M_PI/180);
1546.           gps.latitude = lati;
1547.           gps.longitude = longi;
1548.     return gps;
1549.     }
1550.     void thread_Actualiza_GPS(void)
1551.     {
1552.     char  sResult[200];
1553.     TRAMA_GPS gps1;
1554.     double  lat_km = 0;
1555.     double  lon_km = 0;
1556.     double lat_final[3] = {41,27,06.54}; //Latitude
1557.     double lon_final[3] = {8,17,38.58}; //Longitude
1558.     double lon_f,lat_f,aux2;
1559.     double dx,dy,dt;
1560.     double bussola;
1561.     float erroBGPS = 0.00;
1562.           aux2 = lat_final[0] + lat_final[1]/60 + lat_final[2]/3600;
1563.           lon_f = lon_final[0] + lon_final[1]/60 +
lon_final[2]/3600;
```

```
1564.          lat_f = 6378.0*sin(aux2*M_PI/180.0);
1565.          lon_f = 6378.0*cos(aux2*M_PI/180.0)*sin(lon_f*M_PI/180.0);
1566.          while(a_correr) //Depois tenho q por a ver se esta a cerca
de 5m do ponto, se sim, passa para um proximo ponto
1567.          {
1568.                if(GPS == 0)
1569.                {
1570.                      usleep(500);
1571.                      fcntl(fdGPS, F_SETFL, FNDELAY); // don't block
serial read
1572.                      if (ReadAdrPort(sResult,200) > 0)
1573.                      {
1574.
   //$GPRMC,130950,A,4127.1007,N,00817.5508,W,000.0,194.6,191007,004.1,W*
79
1575.                            if(!strncmp(sResult,"$GPRMC",6) &&
sResult[14] == 'A')   //atof converte string to float
1576.                            {
1577.                                  gps1 = converte_Km(sResult);
1578.                                  lat_km = gps1.latitude;
1579.                                  lon_km = gps1.longitude;
1580.                                  // considerando x a longitude  e y
a latitude temos em xy
1581.                                  dx = lon_f - lon_km;
1582.                                  dy = lat_f - lat_km;
1583.                                  dt = ((sqrt(dx*dx +
dy*dy))*1000.0);
1584.                                  bussola = 49999.999;
1585.                                  if(dt == 0)
1586.                                        bussola = 0;
1587.                                  else
1588.                                  if(dy == 0)
1589.                                        if(dx > 0)
1590.                                              bussola = 0;
1591.                                        else
1592.                                              bussola = 180;
1593.                                  else
1594.                                  if(dx == 0)
1595.                                        if(dy > 0)
1596.                                              bussola = 0;
1597.                                        else
```

```
1598.                                         bussola = 180;
1599.                                   else
1600.                                   if(dx > 0)
1601.                                         if(dy > 0)
1602.                                         {bussola = 360 -
atan(dx/dy)*180.0/M_PI;}
1603.                                         else
1604.                                         {bussola = 180.0 -
atan(dx/dy)*180.0/M_PI;}
1605.                                   else
1606.                                         if(dy > 0)
1607.                                         {dx = dx * (-1);

1608.                                         bussola =
atan(dx/dy)*180.0/M_PI;}
1609.                                         else
1610.                                         bussola = 180.0 -
atan(dx/dy)*180.0/M_PI;     // ok
1611.                                   bussola = (bussola * 255)/360;
1612.                                   erroBGPS = bussola - Bussola;
1613.                                   if(erroBGPS > -10 && erroBGPS <
10)
1614.                                   {
1615.                                         //Segue em frente

1616.                                         dirG = 0;
1617.                                   }
1618.                                   if (erroBGPS > 10)
1619.                                   {
1620.                                         //virar para a direita
1621.                                         if(dirG > -80) //mÃ¡ximo de
direcÃ§Ã£o
1622.                                         {
1623.                                         dirG = dirG - 40;
1624.                                         }
1625.                                   }
1626.                                   if(erroBGPS < -10)
1627.                                   {     //virar para a esquerda
1628.                                         if(dirG < 80) //mÃ¡ximo de
direcÃ§Ã£o
```

```
1629.                                         {
1630.                                               dirG = dirG + 40;
1631.                                         }
1632.                                   }
1633.                                   if(dt > 10)
1634.                                   {
1635.                                                     //Desactivar
ComunicaÃ§Ã£o com o PC
1636.                                         printf("\nControlo PC
Desligado\n");
1637.                                         Flag_comunica1 = 1;
1638.                                         limita_vel(dirG,vel);
//actualizar o valor da direccao
1639.                                   }
1640.                                   else
1641.                                   {
1642.                                         //Activar ComunicaÃ§Ã£o com
o PC
1643.                                         printf("\nControlo PC
Ligado\n");
1644.                                         Flag_comunica1 = 0;
1645.                                   }
1646.                                   sscanf(sResult,
"%*41s%5s",velGPS);
1647.                                         velGPS1 = atof(velGPS)*1.88;
1648.                             }//if($GPRMC
1649.
   //$GPGGA,130950,4127.1007,N,00817.5508,W,2,07,2.2,212.7,M,52.0,M,,*5B
para saber o numero de satelites
1650.                             if(!strncmp(sResult,"$GPGGA",6))//
__
1651.                             {
1652.                             sscanf(sResult, "%*41s%2s",nsat);
1653.                             }
1654.                             //$PGRME,5.7,M,8.8,M,10.6,M*1B Para
saber o erro em metros
1655.                             if(!strncmp(sResult,"$PGRME",6))
1656.                             {
```

```
1657.
   sscanf(sResult,"%*7s%f%*3s%f%*3s%f",&erroX,&erroY,&erroZ);
1658.                              }
1659.                              //$PGRMT,GPS 18PC board ver. 7 -
software ver. 2.30,P,P,R,R,P,,28,R*56 //temperatura entre outros
1660.                              if(!strncmp(sResult,"$PGRMT",6))

1661.                              {
1662.                              sscanf(sResult,"%*61s%2s",tempGPS);
1663.                              }
1664.                        }
1665.                        else        //Se Erro porta ou GPS Desligado
1666.                        {
1667.                              printf("\t\t\t\tGPS OFF Erro Porta\r");
1668.                        }
1669.                  }
1670.            }//end while
1671.      }
1672.      //Percorre todo o Barramento i2c actualizando os valores dos
Sensores
1673.      //e fazenso andar os motores
1674.      void thread_i2c(void)
1675.      { int i=299;
1676.            printf("\nInicio da thread i2c\n");
1677.            while (a_correr)
1678.            {
1679.                  LerBusola();
1680.                  if(ultrasons==1)
1681.                  {
1682.                        Range_Sonar(0xE0,0xE1,0x00,0x03,0x51);
1683.                  }
1684.                  anda(velocidadeD,velocidadeE);
1685.                  if(print_v == 1)
1686.                  {
1687.                        print_variaveis(vel,dir);
1688.                  }
1689.                  i++;
1690.                  if(i == 300)
1691.                  {
1692.                        Calcula_Humidade();
1693.                        LerTem_Current();
```

```
1694.                       i = 0;
1695.                 }
1696.           }
1697.     }
1698.     int main(void)
1699.     {
1700.     char c;
1701.     pthread_t thread_das_leituras;
1702.     pthread_t thread_do_encoder_E;
1703.     pthread_t thread_do_encoder_D;
1704.     pthread_t thread_do_Joystic;
1705.     pthread_t thread_da_comunicacao;
1706.     pthread_t thread_da_ACELX;
1707.     pthread_t thread_da_ACELY;
1708.     pthread_t thread_dos_Fusiveis;
1709.     pthread_t thread_do_GPS;
1710.           le_config("REDE.txt");
1711.           le_config("CONTROLO.txt");
1712.           if (abrir_i2c()==1)
1713.           {
1714.                 printf("\n\nErro\r");
1715.                 exit(0);
1716.           }
1717.     //Pinos
1718.           if ((fdPin = open("/dev/gpiog", O_RDWR))<0)
1719.           {
1720.                 printf("\n\n\n\n\n\n\n\nOpen error on
/dev/gpiog\n");
1721.                 return 0;
1722.           }
1723.     //JOYSTIC
1724.           fd = open("/dev/ttyS2", O_RDWR | O_NOCTTY | O_NDELAY);
1725.           if (fd == -1)
1726.                 {
1727.                       perror("open_port: Unable to open /dev/ttyS2 -
\r");
1728.                       return 0;
1729.                 }
1730.                 else
1731.                 {
1732.                       fcntl(fd, F_SETFL, 0);
```

```
1733.                }
1734.    //GPS
1735.          OpenAdrPort("0");
1736.          print_menu();
1737.          md03_i2c(MOTOR_ESQ,0x03,ACEL);      //Aceleracao
1738.          md03_i2c(MOTOR_DIR,0x03,ACEL);      //Aceleracao
1739.          pthread_create(&thread_das_leituras, NULL, (void
*)thread_i2c, NULL);
1740.          pthread_create(&thread_do_encoder_E, NULL, (void
*)thread_encoderE, NULL);
1741.          pthread_create(&thread_do_encoder_D, NULL, (void
*)thread_encoderD, NULL);
1742.          pthread_create(&thread_do_Joystic, NULL, (void
*)thread_joystic,NULL);
1743.          pthread_create(&thread_da_comunicacao, NULL, (void
*)thread_comunica,NULL);
1744.          pthread_create(&thread_da_ACELX, NULL, (void
*)thread_AcelX,NULL);
1745.          pthread_create(&thread_da_ACELY, NULL, (void
*)thread_AcelY,NULL);
1746.          pthread_create(&thread_dos_Fusiveis, NULL, (void
*)thread_Actualiza_Fusiveis,NULL);
1747.          pthread_create(&thread_do_GPS, NULL, (void
*)thread_Actualiza_GPS,NULL);
1748.          startjoystic();         //inicializaÃ§Ã£o dos parametros
do joystic
1749.    while(a_correr)
1750.          {
1751.                c=mygetch();
1752.                switch(c)
1753.                {
1754.                      case 'q': //aumentar a velocidade
1755.                            if (vel+20<=255)
1756.                            vel=vel+20;
1757.                            //vel=50;
1758.                            break;
1759.                      case 'a': //diminuir a velocidade
1760.                            if (vel-20>=-255)
1761.                            vel=vel-20;
1762.                            break;
```

```
1763.                       case 'o': //aumentar a direccao virar para a
esquerda
1764.                             dir=dir+10;
1765.                             break;
1766.                       case 'p': //diminuir a direcÃ§Ã£o virar para a
direita
1767.                             dir=dir-10;
1768.                             break;
1769.                       case 'z':
1770.                             ACEL+=10;
1771.                             md03_i2c(MOTOR_ESQ,0x03,ACEL);
   //Aceleracao
1772.                             md03_i2c(MOTOR_DIR,0x03,ACEL);
   //Aceleracao
1773.                             break;
1774.                       case 'x':
1775.                             ACEL-=10;
1776.                             md03_i2c(MOTOR_ESQ,0x03,ACEL);
   //Aceleracao
1777.                             md03_i2c(MOTOR_DIR,0x03,ACEL);
   //Aceleracao
1778.                             break;
1779.                       case 'l':
1780.                             vel = 200;
1781.                             break;
1782.                       case 'j':
1783.                       case 'J':
1784.                             Flag_stop_joystic=!Flag_stop_joystic;
1785.                             if(Flag_stop_joystic == 0 )
1786.                                   {printf("\nJOYSTIC LIGADO\n");}
1787.                             else {printf("\nJOYSTIC DESLIGADO\n");}
1788.                             break;
1789.                       case 'c':
1790.                             Flag_comunica=!Flag_comunica; //Activa
ComunicaÃ§Ã£o controlo autonomo Imagem
1791.                             break;
1792.                       case 'u':
1793.                             ultrasons=!ultrasons; //Activa ultrasons
1794.                             break;
1795.                       case 'm':
```

```
1796.                         flagcontrolo=!flagcontrolo; //Activa
controlo
1797.                         break;
1798.                   case 'g':
1799.                         GPS =! GPS; //Activa GPS
1800.                         break;
1801.                   case 'n':
1802.                         Flag_Notas =! Flag_Notas; //Grava em
ficheiro
1803.                         break;
1804.                   case 'y':
1805.                         Flag_cala_t =! Flag_cala_t; //Desactiva
controlo c o pc
1806.                         break;
1807.                   case ' ':
1808.                         vel=0; dir=0;
1809.                         velD = 0;
1810.                         velE = 0;
1811.                         velocidadeD=0; velocidadeE=0;
1812.                         velocidade_E = 0, velocidade_D = 0;
1813.                         break;
1814.                   case 'b':
1815.                         md03_i2c(MOTOR_ESQ,0x02,0);
1816.                         md03_i2c(MOTOR_ESQ,0x00,0);
1817.                         md03_i2c(MOTOR_DIR,0x02,0);
1818.                         md03_i2c(MOTOR_DIR,0x00,0);
1819.                         vel = 0; dir = 0;
1820.                         break;
1821.                   case 'v':
1822.                         print_v =! print_v; //Mostra VariÃ veis
1823.                         break;
1824.                   case 27:
1825.                         vel=0; dir=0;
1826.                         velD = 0;
1827.                         velE = 0;
1828.                         velocidadeD=0; velocidadeE=0;
1829.                         a_correr=0;
1830.                         break;
1831.                   default:
1832.                         break;
1833.             }
```

L

```
1834.                 limita_vel(dir,vel);
1835.           }
1836.           i2c_close();
1837.           CloseAdrPort();   //Fechar o GPS
1838.           CloseJAdrPort(); //Fechar o JOY
1839.     return 0;
1840.     }
```