PROJECTO FINAL DE CURSO

Sistema de Navegação em Alto Nível para um Robot Móvel em Ambientes Estruturados

Trabalho realizado no âmbito de um projecto do 5º ano
do curso de licenciatura em Engª. Electrónica e Telecomunicações por:

Anabela Almeida Duarte
Paulo Jorge F. Peixoto

Sob orientação de:

Prof. Dr. Vítor Santos (Dep. Engenharia Mecânica)
Prof. Dr. Luís Seabra Lopes (Dep. Electrónica e Telecomunicações)

UNIVERSIDADE DE AVEIRO
Departamento de Electrónica e Telecomunicações
Secção Autónoma de Engenharia Mecânica

Julho 1999

Tabela de conteúdo

Resumo

"... Navegação é a ciência (ou a arte) de dirigir o curso de um robot móvel à medida que atravessa o ambiente (terra, mar ou ar)".

Philip McKerrow

O trabalho apresentado neste relatório descreve a construção de um sistema de navegação para um robot móvel. De facto, se um ambiente é conhecido então deveria ser possível programar a trajectória de um robot com relativa facilidade, o problema são os erros de odometria (erros acumulados durante o movimento) os quais impedem a correcta execução de uma trajectória.

A navegação com base simplesmente na odometria é suficiente para troços curtos tornando-se no entanto inviável para percursos maiores, sendo assim necessário extrair informação do ambiente para corrigir ou suplantar erros odométricos, ou seja, calibrar o robot. Em geral, usam-se referências específicas que necessitam de sensores extra (códigos de barras, lasers, GPS, cameras de vídeo ...). Aqui todo o trabalho assenta numa calibração sem nenhuma referência externa específica mas unicamente com a informação *on-board*.

A ideia fundamental é obter um conjunto de estruturas e ferramentas, estando algumas delas já desenvolvidas, que permitam construir uma missão de navegação num ambiente estruturado.

Assim, ao longo de todo o trabalho não nos preocuparemos com a localização exacta do robot , pois para o fazermos seria imprescíndivel que o ambiente se mantivesse estático e que tivéssemos total confiança quer nas medidas dos ultra-sons quer no sistema odométrico. Para que o sistema seja o mais flexível possível dentro do ambiente que à partida conhecemos, procuraremos implementar algoritmos que alternadamente monitorizem o sistema odométrico e as medidas sensoriais obtidas, tirando o máximo partindo do ambiente, isto é, escolhendo para pontos de calibração referências do ambiente (*landmarks* naturais).

1. Introdução

1.1 . Objectivos

O objectivo principal deste trabalho consiste na concepção de um sistema de definição de missões de navegação de um robot móvel em ambientes conhecidos com base em permissas de alto nível.

O sistema final concebido deverá permitir a um operador especificar uma missão, não apenas em termos de ângulos e distância pois estes acabarão por falhar mais tarde ou mais cedo, mas também num conjunto de condições sensoriais e acções associadas.

Cada missão especificada será um conjunto de comandos que pertencem a um '*Instruction Set*' pré-definido que serão executados sequencialmente e que por sua vez executam as operações de movimento e reconhecimento do meio envolvente do robot. Os comandos serão funções compostas por instruções básicas de locomoção, ou leitura sensorial no robot, que cumprirão um determinado fim durante um determinado período de tempo.

1.2. Estrutura do relatório

Este relatório está dividido em 3 partes principais. Uma primeira parte onde se relatam algumas experiências consideradas relevantes, bem como aspectos relacionados com a familiarização com o robot. A segunda parte apresenta o trabalho própriamente dito, desenvolvido, nomeadamente parametrização e métodos exactos para resolver os respectivos algoritmos e uma descrição do que foi feito. Por fim uma última parte onde descrevemos o que pensamos que poderá ser melhorado, as conclusões sobre o trabalho desenvolvido, bem como a aplicação e desenvolvimento deste trabalho noutras perspectivas que na nossa opinião achamos relevantes.

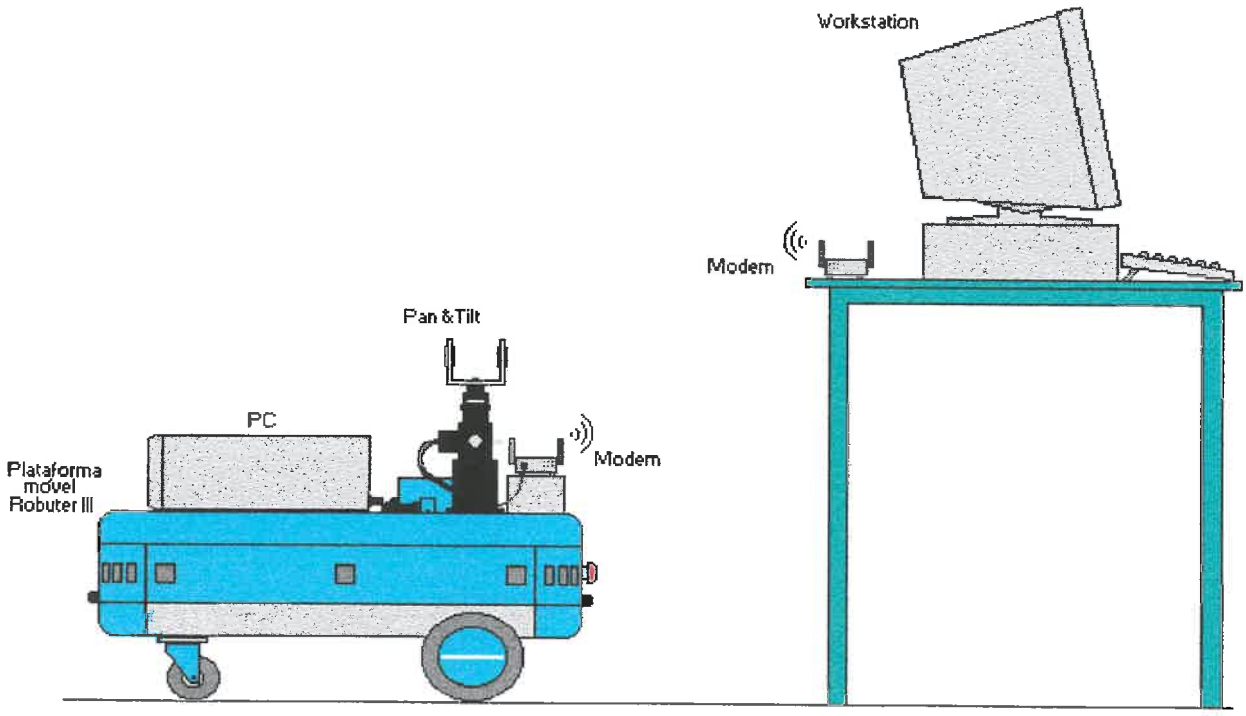
Em complemento apresenta-se ainda um conjunto de anexos que incluem: *Quick User's Guide for Robuter,* que tem por objectivo uma consulta rápida para uma primeira abordagem com o sistema (Anexo I), conjunto de mensagens interpretadas pelo sistema que servem para comunicar em tempo real com o robot à medida que este cumpre uma missão (Anexo II), um manual de edição e gestão de tarefas (Anexo III), e por fim todo o código desenvolvido (Anexo IV).

2. Descrição do ambiente de trabalho

2.1 . Local e Equipamento

Este trabalho foi desenvolvido no âmbito de um projecto final de curso com um perfil de computadores e programação, proposto em conjunto como Departamento de Electrónica e Telecomunicações e a Secção Autónoma de Engenharia Mecânica, sendo no entanto totalmente realizado no Laboratório de Automação e Robótica da Seçcão Autónoma de Engenharia Mecânica.

O equipamento relacionado com o presente trabalho encontra-se esquematizado na figura seguinte e consiste essencialmente numa plataforma móvel do tipo *Robuter* (*Robuter III* da *Robosoft, SA*), uma *Workstation* do tipo *Sun Sparcstation* e um PC acoplado ao robot com o qual comunica por porta série e paralela (apenas para *download*). Este PC está ligado à rede informática por rádio-modem *Ethernet*, podendo também comunicar via cabo directamente com a *Workstation*.



2.2. Especificações do Robot

O robot possui um sistema diferencial de condução baseado em dois motores independentes (2×300 Watts, 48Volts), 4 baterias 12 Volts 60Ah, *encoders* ópticos para posicionamento, uma rede de 24 sensores ultrasónicos mais dois montados numa unidade *Pan & Tilt*, 2 sensores de colisão (*bumpers*), um CPU *Motorola* 68040, o sistema operativo em tempo real *Albatros™*. Pesa 150 Kg (incluindo baterias), pode carregar até 120 Kg e tem aproximadamente 80 cm de altura (com a unidade *Pan & Tilt*) , 78 cm de largura e 1m de comprimento. A sua velocidade de locomoção pode variar desde 5 cm/s até 1 m/s (0.18 Km/h até 3.6 Km/h).

2.3. Plataformas de Software e Linguagens de Programação

Os ambientes de programação consistem em *UNIX* do lado da estação de trabalho e o sistema operativo de tempo real ***Albatros™*** no robot.

A *Workstation* encontra-se equipada com um *cross-compiler Gnu GCC 2.8.x*, estando o PC que se encontra a bordo do robot equipado com *Linux Debian 1.3.1*.

Todo o código desenvolvido (em PC ou na *Workstation*) foi desenvolvido utilizando a linguagem de programação C, devendo ser *cross-compiled* para depois ser transferido para o robot e posto em execução de forma manual através do monitor do próprio robot.

3. Familiarização e adaptação ao sistema

3.1. Algumas experiências com o robot

Este conjunto de experiências serviu para além de nos ajudar a familiarizar com o sistema, para concluirmos à cerca da maneira de actuar no robot para este poder executar um conjunto de movimentos de locomoção básicos.

Todas as experiências foram feitas utilizando pequenos programas os quais impunham movimentos através da invocação de comandos do *Albatros™*.

3.1.1. Erros odométricos

Não foi necessário muito tempo nem muitas experiências para verificarmos que existia uma dessincronização entre o sistema intrínseco de medição de deslocamentos e os deslocamentos efectuados na realidade. Podemos facilmente explicar o problema se pensarmos que o robot pode, por exemplo, deslizar devido às irregularidades do pavimento ou a efeitos de derrapagem ou patinagem das rodas. Contudo, o mais grave neste tipo de erros, é o facto de serem cumulativos.

Uma simples experiência mostra que ao fim de algum tempo existe um desfasamento completo entre as coordenadas reais e as medidas no sistema do robot. Este tipo de experiência pode ser feita por exemplo, fazendo as medidas externamente, ou seja, como podemos ver em [Santos 95] este problema só poderá ser resolvido recorrendo a outros sensores sem serem necessáriamente sensores destinados a medir grandezas de forma cumulativa e absoluta, por exemplo sensores de ultra-sons.

Todo o trabalho desenvolvido assenta no príncipio de que existem erros odométricos, no entanto apresentamos de seguida algumas técnicas que têm sempre como objectivo minimizar este tipo de erros, para que sempre que seja necessário basearmo-nos na odometria esta seja o mais fiável possível. Relembramos ainda que o recurso à odometria é necessário, no entanto será ao longo de todo o trabalho reduzido ao minimo.

3.1.2. Erros dos sensores de ultra-sons

Os problemas associados aos sensores de ultra-sons são sobejamente conhecidos e questões relacionadas com a interferência entre sensores e reflexões especulares no ambiente comprometem a interpretação das medidas efectuadas, bem como o facto de que aproximadamente uma em cada 10 medidas falhar. Este valor é resultado de outras experiências realizadas com uma plataforma com o mesmo tipo de sensores em [Santos 95].

Em relação a interferências não registámos nenhum caso, devido aos sensores da plataforma serem disparados por nodos fisicamente separados. Isto é, o sistema é dotado de 2 planos de simetria, existindo 6 grupos de 4 sensores ortogonais nas suas direções sendo assim possível disparar os 24 sensores por 6 vezes com um risco quase nulo de interferência. Quanto às reflexões especulares estas ocorrem quando as dimensões dos "corpos" reflectores ultrapassam o comprimento de onda do feixe incidente, neste caso o ângulo de reflexão é igual ao ângulo de incidência (Lei de Snell), isto é, corre-se o risco do feixe reflectido não encontrar o respectivo sensor.

Após alguns testes verificámos que os valores medidos pelos sensores de ultra-sons são superiores aos reais sendo constante o erro de que as medidas são afectadas.

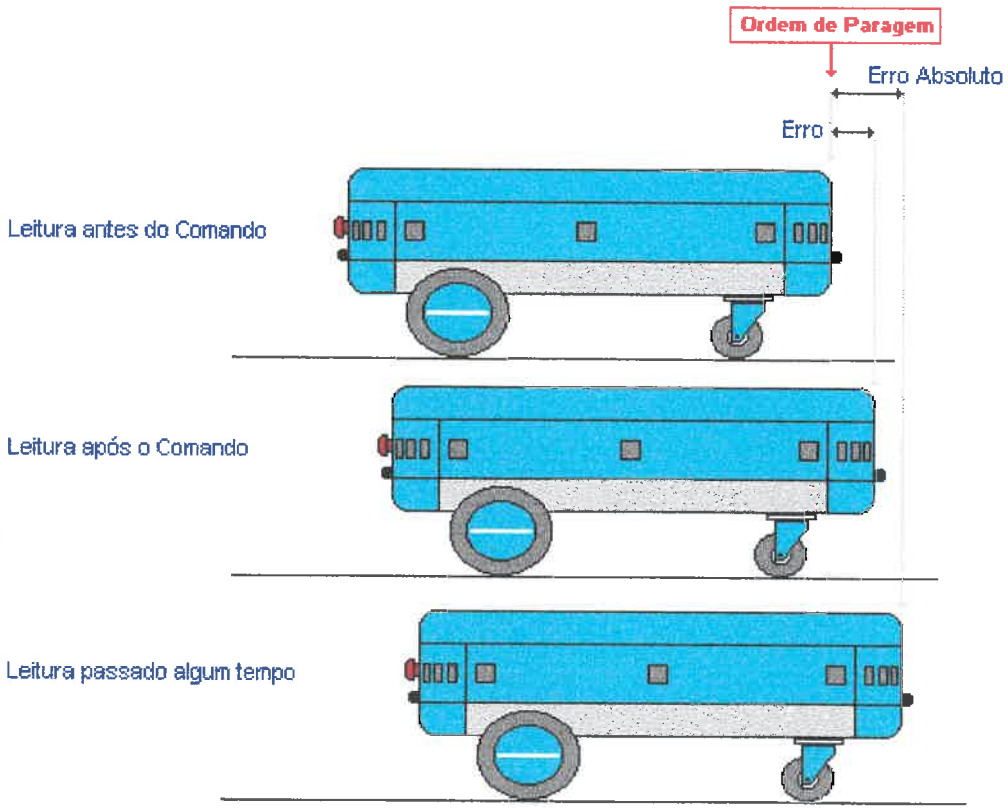
Tendo em conta o príncipio físico em que se baseiam, que consiste em medir o tempo que decorre entre a emissão de uma onda e a recepção de um eco devido à reflexão em "corpos" na trajectória do feixe incidente e sabendo a velocidade de propagação do som é imediato deduzir qual a distância mais curta ao ambiente reflector na direcção do feixe emitido.

Para melhorar os resultados obtidos diminuímos o valor da velocidade do som, usado pela função READ do ***Albatros™***, conseguindo assim valores mais próximos das distâncias reais. Verificamos no entanto que o valor da velocidade do som já se encontrava optimizado no trabalho desenvolvido anteriormente [Ranger98].

3.1.3. Imobilização do robot

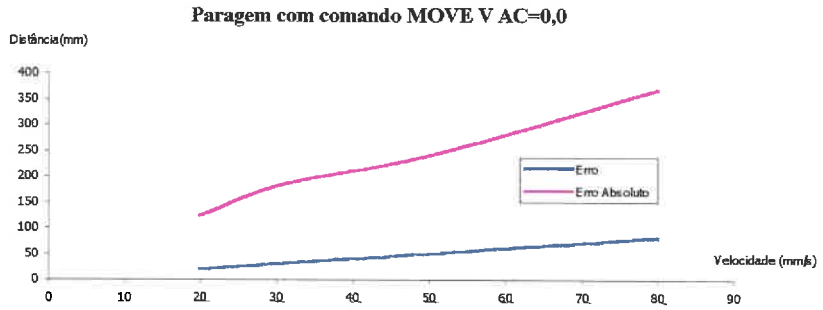
Existem duas maneiras de imobilizar o robot: uma que actua a baixo nível cortando a corrente dos motores impedindo qualquer outro movimento de actuar que é o comando *serv off* do ***Albatros™***, o qual só depois de fazer *serv on* permite reinincializar qualquer outra acção de movimento; a outra que actua a mais alto nível e que impõe simplesmente uma velocidade igual a zero nas rodas que se faz utilizando o comando *move v ac=0,0*.

Para podermos decidir qual dos comandos a utilizar em caso de querer imobilizar o robot impusemos um movimento uniforme ao robot (durante 5 metros) e medimos a diferença entre o lugar onde o robot deveria parar e onde parava efectivamente, como mostra a seguinte figura.



Repetimos a experiência com o comando *serv off* e com o comando *move v ac=0,0* para diferentes velocidades e verificámos que os erros associados à orientação do robot são mais significativos com o comando *serv off*.

Assim, concluímos que o uso do comando *move v ac=0,0* se torna sempre mais vantajoso em relação ao comando *serv off*. Salientamos também o facto do erro ser proporcional à velocidade como mostra o gráfico seguinte:



Em relação ao comando *serv off* não poderemos construir um gráfico idêntico uma vez que na versão do sistema operativo que o robot tem, quando executamos o comando *serv off*, a odometria é desligada. Podemos, mesmo assim afirmar que a desvantagem é obvia no que diz respeito ao desalinhamento, pois este é mais visivel neste comando através de observação exterior ao robot.

3.2. Familiarização com o trabalho já desenvolvido

3.2.1. Introdução

Como já foi mencionado anteriormente existe já desenvolvido um conjunto de funções de baixo e médio nível para navegação do robot e que usa essencialmente dados vindos dos sensores de ultra-sons.

O trabalho desenvolvido é constituído por 5 módulos de baixo nível que implementam a comunicação, o sistema sensorial, o sitema de locomoção e o sistema de emergências. Existem ainda mais dois módulos a nível mais superior: um que acrescenta a funcionalidade de mais dois sensores montados numa unidade *Pan & Tilt* e outro módulo que implementa o atravessamento de passagens estreitas.

3.2.2. Descrição sumária dos módulos

3.2.2.1. *kernel.c*

Este módulo é o responsável pelo lançamento de todos os processos que constituem o sistema. Assim começa por invocar todas as funções de inicialização de processos lançando cada processo com os respectivos período, prioridade e off-set, conforme a tabela *proc to add* [Robosoft97b], [Robosoft97c]. De seguida fica em ciclo infinito enquanto a variável de controle *kill* tiver o valor *FALSE*. Quando se pretende remover a tabela de processos coloca-se a variável *kill* a *TRUE*, interrompendo o ciclo infinito do módulo *kernel.c*. Quando se pretende

adicionar um outro módulo ao sistema, este deve ser incluido na tabela de processos que se encontra neste módulo, atribuindo-lhe o período, a prioridade e o *off-set* adequados.

3.2.2.2. *serial.c*

(Nota: A descrição seguinte é referente à versão implementada no trabalho anterior)

Este é o módulo responsável pela comunicação com o exterior através da porta série. Tem implementado um protocolo com 3 categorias de mensagens:

Tipo A – Envio de comandos;
Tipo B – Pedido de informação ao robot;
Tipo U – Mensagens associadas aos ultra-sons.

Para além de testar se existem mensagens este módulo trata-as, seja através do envio da respectiva resposta ou da actualização dos parâmetros adequados.

O formato utilizado para as mensagens é baseado sobre pesquisas anteriores desenvolvidas na área de comunicação com plataformas móveis.

Neste formato as mensagens são constituídas pela categoria, representado pelo primeiro caracter, a seguir vem o tipo de mensagem e por fim, os parâmetros quando necessários.

Uma descrição mais pormenorizada das várias categorias de mensagens encontra-se no Anexo II. Estas mensagem podem ser enviadas durante a execução do programa de missão.

3.2.2.3. *us.c*

Módulo responsável pela gestão dos sensores de ultra-sons. Podem-se configurar da maneira mais adequada todos os sensores, estando sempre dísponiveis todos os dados relativos a estes, não sendo necessário qualquer procedimento de baixo nível para interface com os sensores (excepto em situações especiais).

Todas as variáveis estão encapsuladas numa estrutura do tipo **US_DATA** cujo nome é ***sens*** e da qual destacamos duas variáveis internas: ***active_sensors*** e ***values*** que são respectivamente a tabela de sensores activos e a tabela com os respectivos valores de cada sensor.

3.2.2.4. *executor.c*

Este módulo é o responsável pela imposição de velocidade nas rodas e leitura dos odómetros. Num dado instante apenas uma dada velocidade estará a ser imposta em cada roda, pois apenas este módulo o pode fazer, ficando assim eliminadas situações de instabilidade devido a vários módulos imporem velocidades diferentes nos motores. Assim, sempre que se quiser impôr velocidade nos motores esta acção terá que ser feita através da estrutura que este módulo detém para o efeito (variável ***mov*** do tipo **MOVE**) que contém a velocidade de cada roda (left e right). Por outro lado este módulo faz a leitura constante dos odómetros estando sempre disponível o valor de x, y e θ na variável ***pos*** do tipo **POSTURE**.

3.2.2.5. *emerg.c*

Módulo responsável pela imobilização do robot em situações de emergência sendo consideradas como situações de emergência a aproximação crítica a objectos, a colisão ou a perda de comunicações. Este módulo monotoriza o valor dos ultra-sons e as velocidades que estão a ser impostas nas rodas concluindo se existe ou não aproximação crítica a um objecto.

O conjunto de sensores escolhidos é calculado através da trajectória que o robot tem, havendo um sensor central que é escolhido analisando a velocidade de cada roda. É também analisada a ocorrência de colisões ao monotorizar o estado dos bumpers, pois caso estes estejam activados é sinalizada uma *flag* de emergência. Esta situação só pode ocorrer se o sistema baseado nos sensores falhar o que é possível devido às próprias limitações destes sistemas.

Por último existe um teste de perda de comunicação que é feito com base no *time stamp* da última mensagem recebida e do tempo do relógio do robot. Assim, se não houver uma comunicação a menos de 3 minutos a flag de emergência é activada.

A imobilização do robot é efectuada através do comando *serv off* feito logo que seja detectada por este módulo uma situação de emergência.

O módulo disponibiliza a variável ***emerg*** do tipo **EMERGENCY** que contém todas as *flags* deste módulo.

Um dos requesitos impostos por este módulo é que cada vez que se pretenda alterar a configuração dos sensores activos (ver módulo *us.c*) é necessário colocar a *TRUE* a variável ***emer.us_change*** para que o módulo de emergêngia permita ou não a actualização da tabela de sensores activos mediante o estado do ***auto-toogle of useless sensors***.

Outro requesito é que para validar um pedido de movimento, ou seja, para que as variáveis ***mov.right*** e ***mov.left*** executem os comandos de velocidade nas rodas pretendidos é necessário colocar a variável ***emerg.rtm*** com o valor *TRUE* para validar esse pedido de movimento.

3.2.2.7. *doors.c*

Este módulo executa o atravessamento de passagens estreitas, sendo para isso activado ou desactivado quando necessário através da variável ***auto_detect***. Durante o atravessamento da passagem estreita todo o movimento é controlado por este módulo. Quando concluído o procedimento é activada a variável ***success***, podendo assim os módulos que o utilizarem monotorizar esta variável afim de desactivarem o módulo para que o fluxo da missão prossiga.

O algoritmo para o atravessamento de passagens estreitas [Ranger98] utiliza funções auxiliares e também o módulo *ptu.c* descrito no item seguinte.

O procedimento começa por invocar a função ***detect_door***, ou seja é detectada a passagem estreita e de seguida é activada a unidade *Pan & Tilt* e desligados os restantes sensores da plataforma. Assim e apenas com base nos dois sensores da *Pan & Tilt* o robot é posicionado o mais central possível relativamente à passagem estreita para que o atravessamento seja possível avançando de seguida para o interior da passagem estreita. Na impossibilidade das distâncias mínimas poderem ser garantidas é sinalizada a impossibilidade do atravessamento.

Referimos o facto deste módulo necessitar ainda de ligeiros refinamentos devido a apresentar alguma instabilidade. Detectámos após algumas experiências que aproximadamente 20% das vezes faz a abordagem errada à passagem estreita podendo mesmo em alguns casos colidir.

3.2.2.6. *ptu.c*

Este módulo faz o interface com a unidade *Pan & Tilt* (onde estão montados 2 sensores diametralmente opostos) comportando algumas funcionalidades para esse efeito. Para a comunicação com outros módulos destacamos as variáveis globais ***ptu_on***, ***pt_values*** e ***d_values***, as quais são actualizadas pelo módulo *doors.c*, descrito anteriormente. A variável *pt_values* contém valores amostrados de 5 em 5 graus, numa rotação completa de 360º, estando assim coberto o espaço envolvente do robot.

O posicionamento da unidade *Pan & Tilt* pode ser alterado podendo o módulo funcionar apenas mudando os parâmetros de posicionamento para esse efeito [Ranger98].

3.2.2.7. Interligação com os novos módulos

No desenvolvimento do projecto, construiram-se duas estruturas diferentes que correspondem a maneiras diferentes de executar uma tabela de tarefas. A primeira é através de uma estrutura do tipo *case*, que a cada estado faz corresponde a execução de uma acção, não sendo permitida qualquer alteração depois de compilado o código fonte. Este tipo de estratégia é executada pela estrutura representada no Diagrama 1.

Na figura seguinte a àrea encerrada pela linha a azul contém os módulos que foram desenvolvidos no trabalho anterior (com as respectivas relações entre estes), o módulo DEMO1 é pertencente ao novo trabalho interligando-se inteiramente nos módulos já existentes, embora cada um respeite completamente a integridade dos outros.

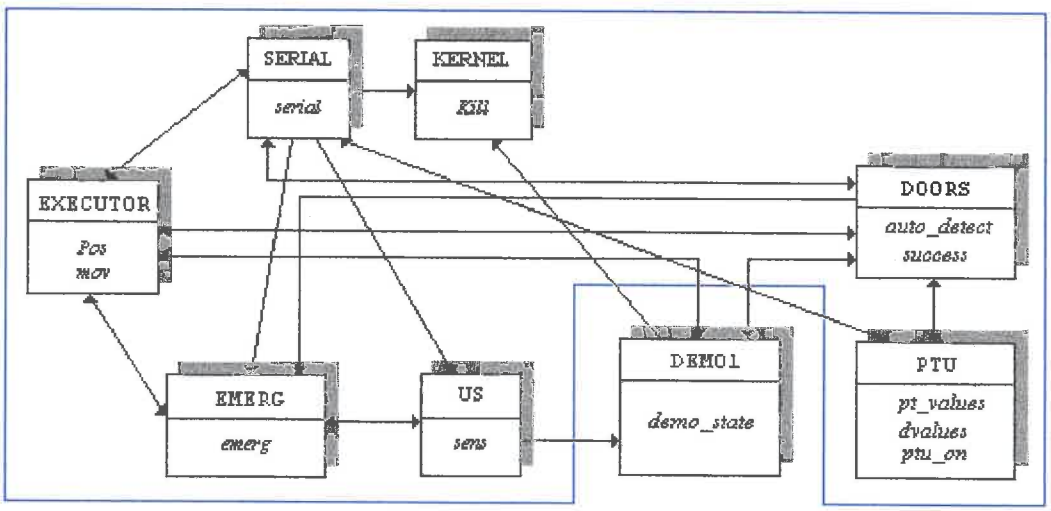


Diagrama 1

O segundo tipo de estratégia implementa um modo interactivo de programar o robot, sendo possivel, no fim de carregado o programa base, programar o robot através de comandos isolados ou fazendo o *download* de um conjunto de tarefas, que juntas, formam uma missão.

Este tipo de estratégia permite uma total gestão da missão, sendo possivel inserir, remover e alterar tarefas em tempo real, ou seja, é possivel por exemplo interromper uma tarefa que esteja a ser executada e alterar uma ou mais tarefas, podendo assim alterar a missão à vontade do

utilizador, sem que o sistema tenha que ser reiniciado,ou que seja necessário recompilar algum código.

O Diagrama 2 representa a estrutura desta estratégia, sendo incluido no diagrama o monitor da porta série, já que agora é uma parte importante do sistema, devido a que, é através dele que o módulo SERIAL comunica com o utilizador. Mais uma vez, toda a integridade de cada módulo foi respeitada, centrando-se o controlo do robot nos módulos MISSION_EXECUTOR e SERIAL que fazem a execução e gestão, respectivamente, do *array* de missões (MISSION_ARRAY).

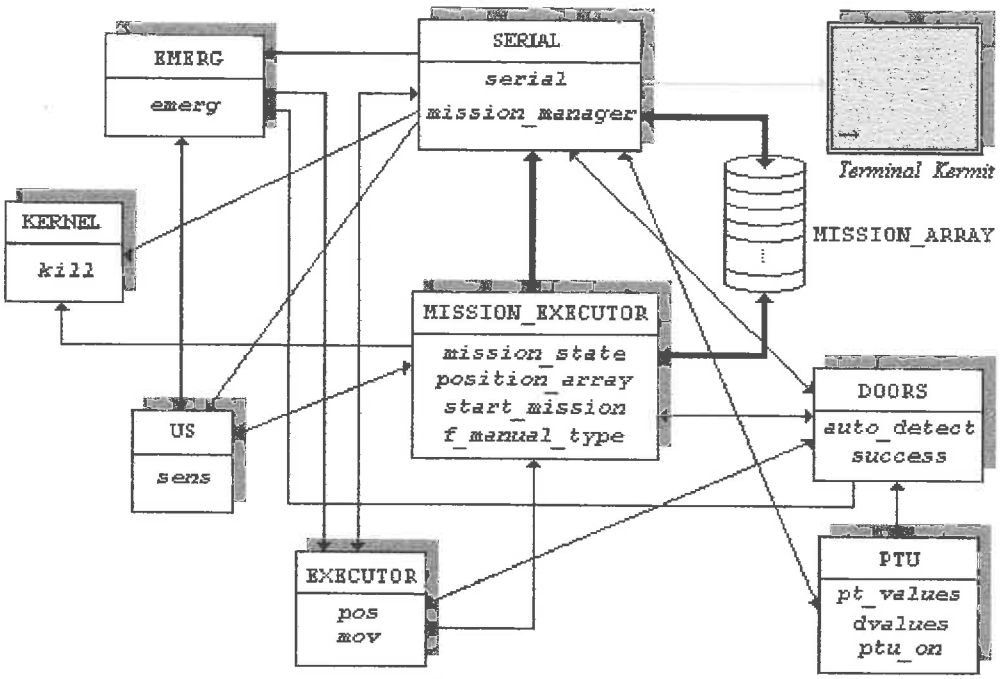


Diagrama 2

Neste fase do projecto, o módulo SERIAL está totalmente reestruturado, permanecendo no entanto as rotinas de comunicação base. O módulo DEMO1 foi substituido pelo módulo MISSION_EXECUTOR que é uma estrutura que, tal como a anterior, comporta todas as rotinas referentes as várias tarefas, mas a execução destas (feitas pela estrutura case anterior), é feita por uma estrutura controlada pelo MISSION_ARRAY e pelo módulo SERIAL, através de variáveis globais.

4. Elementos do trabalho efectuado

4.1. Definiçao de um conjunto de acções e sua parametrização

A definição de uma missão para o robot implica que a trajectória seja decomposta num conjunto de acções o mais atómicas possiveis. Este número limitado de acções deve portanto abranger todo tipo possível de trajectórias que o robot possa efectuar tendo em conta obviamente as dimensões físicas e as limitações de manobra deste.

Assim, caracterizamos as acções em quatro tipos:

→ (Tipo 1) Acções que executam movimentos geometricamente bem definidos que descrevem figuras geométricas conhecidas, das quais configuramos a dimensão através dos seus parâmetros de entrada.

- *roll(θ , v)*
- *go(d , v)*
- *circ(r , θ, v)*
- *curve(l , θ, v)*

→ (Tipo 2) Acções que executam procedimentos mais elaborados como manobras de alinhamento (paralelismo), atravessamento de passagens estreitas e outras.

- *parallel(s)*
- *cross_door( )*

→ (Tipo 3) Acções mistas que embora sejam compostas por algoritmos semelhantes às acções do primeiro tipo descrito, são caracterizadas por tentarem garantir uma determinada condição, ou seja, o objectivo do movimento é garantir que a condição imposta é ou não mantida conforme os parâmetros de entrada, baseando-se no sistema sensorial para a verificação e calibração do movimento.

- *go_until (d , s , v)*
- *go_parallel(d , v , s)*
- *go_at(df , v , s , dw)*

Com a conjugação destas acções podemos construir missões capazes de cumprir os objectivos pretendidos. A decomposição da trajectória é um passo importante em todo o processo, devido a poder ser nesta fase que, através da escolha de um conjunto de tarefas adequadas a cada situação, poder tirar-se o melhor rendimento destas, levando assim à concretização da trajectória pretendida (missão), da maneira mais eficiente.

4.1.1. A acção *go*

A acção ***go*** impõe um movimento rectílineo ao robot durante um determinado espaço com uma determinada velocidade. Assim, esta função aceita como parâmetros de entrada a distância (centímetros) e a velocidade(cm/s) com que queremos percorrer essa distância.

Se pretendemos que sejam percorridas distâncias positivas, ou seja, que o robot ande para a frente fornecemos um valor de velocidade positiva, se pretendemos que o robot ande para trás fornecemos velocidades negativas. O facto de manter o parâmetro distância sempre positivo e actuar na velocidade se queremos o robot a andar para a frente ou para trás respectivamente, prende-se com o próprio conceito de distância apesar de podermos perfeitamente manter o parâmetro velocidade, mudando o sinal da distância e como fácilmente se compreende em nada se afectava o desempenho do algoritmo.

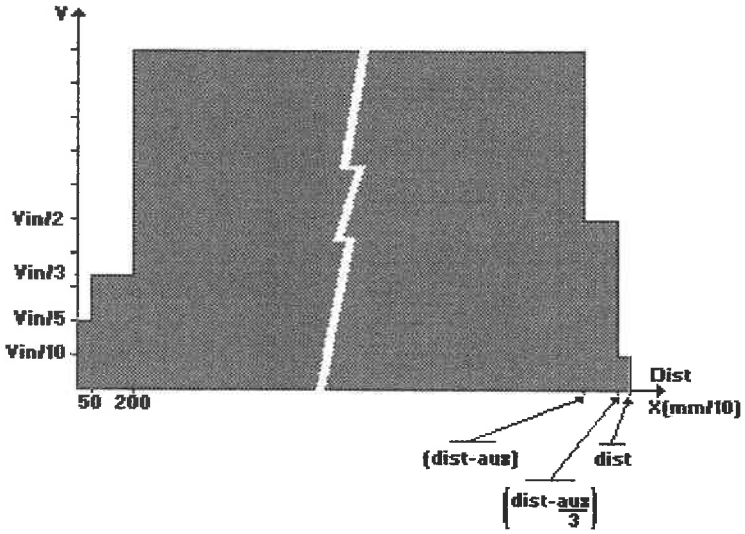
A função ***go*** recorre a uma estrutura do tipo **S_GO** onde se guardam as variáveis internas da máquina de estados e cuja estrutura de dados definimos do seguinte modo:

```
typedef struct      {
      int state;                  /* estado da função g            */
      long int old_x ;            /* valor inicial de x            */
      long int old_y;             /* valor inicial de y            */
      long int old_theta          /* valor inicial de theta        */
      int v;                      /* velocidade média              */
      int erro;                   /* erro associado ao movimento   */
      } S_GO;
```

Esta função, bem como as seguintes, guarda os valores de odometria antes de iniciar o movimento e em seguida coloca-os a zero, quando o movimento termina são somados aos valores actuais os que foram guardados inicialmente, não havendo qualquer erro nos valores de odometria devido a esta operação sobre estas variaveis do sistema. A razão desta opção foi principalmente devido a que, se fossem calculados vectores para concluir (baseando-se na odometria inicial) qual a posição final que o robot teria que ter, para efectuar a tranjectória pretendida, obteria-mos erros erros relativamente elevados. Essa foi a abordagem inicial e numa tentativa de minimizar erros de posição decidimos pela descrita inicialmente.

O problema principal deste tipo de abordagem é que, se as variáveis globais de posicionamento forem consultadas durante a execução da acção *go*, poderão induzir em erro. Este problema é resolvido se, no momento da consulta das variáveis de posicionamento, forem analizados os estados das várias acções residentes. Assim, desde que detectada qual das funções está em execução, basta somar o valor inicial de posicionamento, que cada função guarda na sua estrutura correspondente, obtendo-se o valor verdadeiro de posicionamento global.

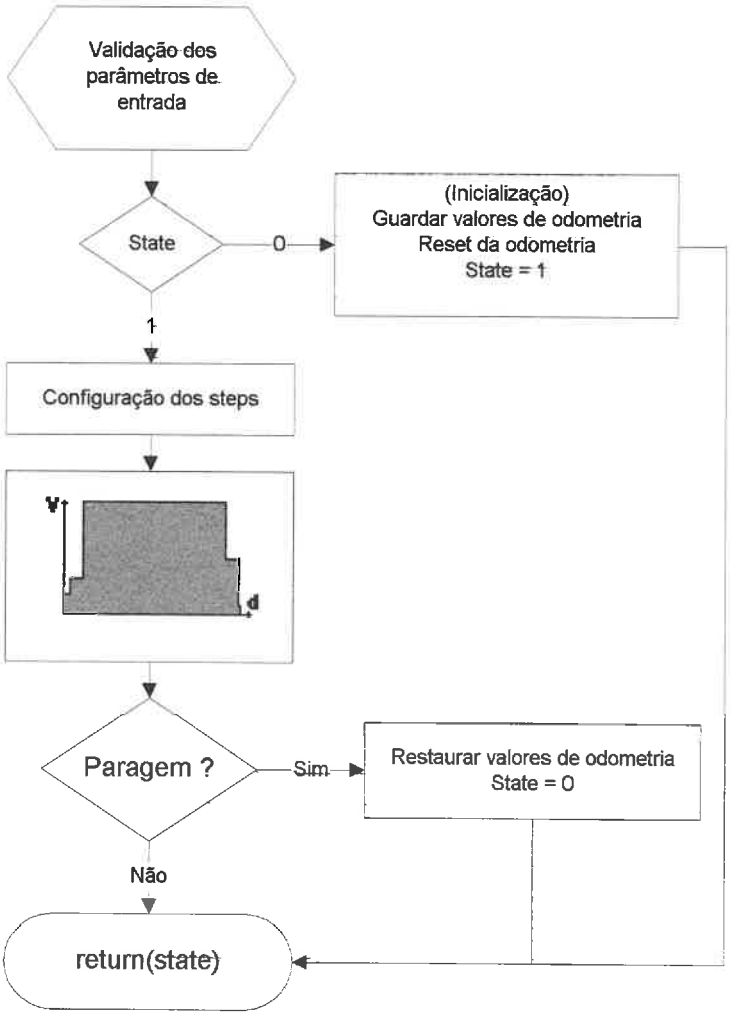
Como resultado de algumas experiências realizadas com o robot constatámos que mudanças bruscas nas velocidades impostas nas rodas faziam com que este se desalinhasse provocando desvios na trajectória, ou até mesmo derrapasse (no caso de o querer parar), o que consequentemente aumentava os erros odométricos. Assim, e com a finalidade de minimizar este facto optámos por impor as velocidade pretendidas por degraus, isto é, antes de passar de um valor de velocidade pretendido impõem-se velocidades intermédias, o que provoca um efeito de (des)aceleração, de acordo com o que podemos observar na figura seguinte.



$$aux = \begin{cases} 3000 & \Leftarrow dist > 20000 \\ 2000 & \Leftarrow dist < 20000 \end{cases}$$

Como a velocidade vai variar na execução da acção *go* a velocidade média efectiva vai ser menor que o valor do parâmetro de entrada, sendo a variável *v* da estrutura associada, que contém o valor da velocidade que está a ser aplicada nas rodas. O gráfico que idealmente seria pretendido, seria uma curva de velocidade sem descontinuidades, com uma aceleração inicial obviamente positiva e negativa na parte final da trajectória.

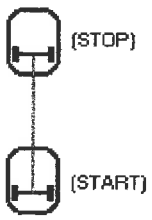
O sistema operativo ***Albatros™***, na versão actual, contém comandos com os quais é possivel implementar movimentos indicando a aceleração desejada. Este tipo de comandos, não pode no entanto ser usado, devido ao facto de não ser possivel a sua coexistência com o modo de comandos em velocidade, com o qual todo o sistema se encontra construido.



Efectuadas algumas experiências verificámos um erro sistemático na distância percorrida em relação à pretendida, assim introduzimos uma pequena correcção interna (obtida por experimentação) que vai compensar esse efeito. Essa correcção é feita através da variável `erro` da estrutura **S_GO** podendo ser alterado o seu valor no início da função.

Na utilização da função no programa principal (que executa a missão) podemos se for necessário, monitorizar o seu estado através da variável *s_go.state* que é devolvida pela função. Nesta versão final apenas existem dois estados (0 e 1) mas facilmente se pode repartir em mais estados se tal for necessário no futuro. Todas as funções seguintes seguem aproximadamente esta filosofia podendo também fazer-se a analogia (em relação à imposição de velocidades) a este fluxograma com as devidas adaptações.

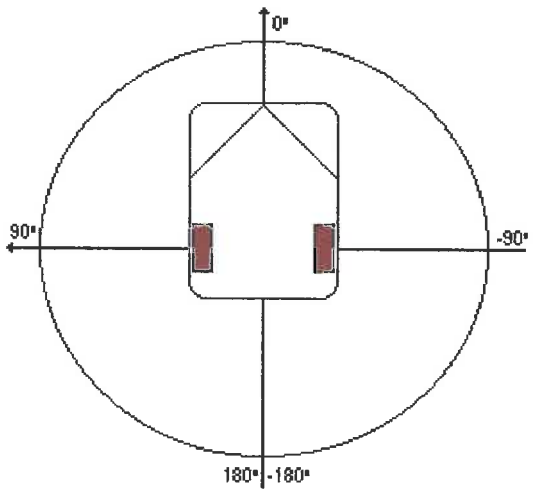
Exemplo de invocação:



go(200,20)

4.1.2. A acção *roll*

A função ***roll*** impõe movimentos simétricos nas duas rodas do robot de tal forma que este rode sobre si próprio um ângulo θ em graus com uma velocidade *v* nas rodas em cm/s, sendo estes os seus parâmetros de entrada: θ e v.



A função ***roll*** recorre a uma estrutura do tipo **S_ROLL** onde se guardam as variáveis internas da máquina de estados e cuja estrutura de dados definimos do seguinte modo:

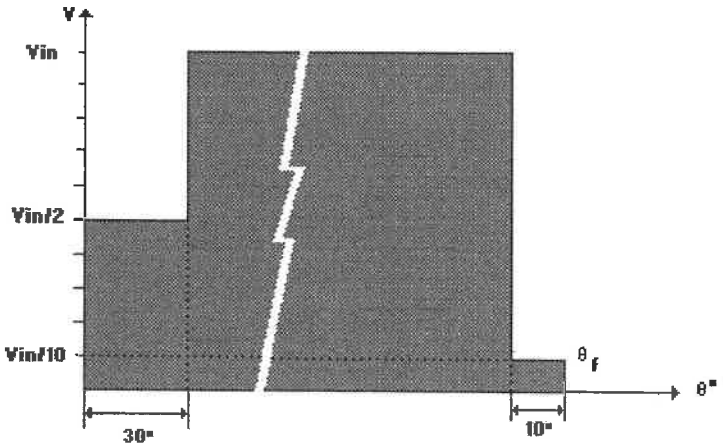
```
typedef struct      {
      int state;                  /* estado da função roll          */
      long int old_x ;            /* valor inicial de x             */
      long int old_y;             /* valor inicial de y             */
      long int old_theta          /* valor inical de theta          */
      int v;                      /* velocidade                     */
      int erro;                   /* erro associado ao movimento    */
      } S_ROLL;
```

Existe uma variável *erro* que diminui o valor absoluto do ângulo de entrada e que serve para fazer uma correção entre o valor do ângulo de rotação pretendido e o medido pela odometria, pois é com base nesta que é testada a condição de paragem. Este valor foi obtido ao fim de alguns testes com a função. Por outro lado, concluimos que a dependência com a velocidade de entrada não se verificava o suficiente para o espectro de velocidades usado, obviamente ela existe mas os resultados obtidos, fixado este valor, eram suficientes para diminuir o erro até valores aceitáveis.

Refira-se o facto de que a diminuição do erro de rotação através da adaptação do valor da variável `erro` comforme os parâmetros de entrada não vai anular o erro final causado por outros factores, alguns não equacionáveis, do ponto de vista da função.

Tal como na função ***go*** a velocidade imposta é feita por degraus até que seja atingido o ângulo pretendido. Assim, enquanto a diferença entre o valor final e o actual for aproximadamente

30° a velocidade é metade da do valor de entrada e quando está próximo do ângulo (quando faltam 10° para atingir o valor final) pretendido, é reduzida para 1/10 do valor da velocidade de entrada , sendo igual a este durante o restante percurso, como podemos visualizar na figura seguinte.



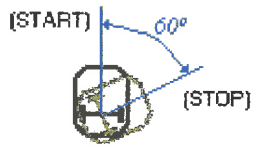
Como podemos observar, o nº de degraus que existe nesta função é diferente do existente na função *go*. Esta opção prende-se com o facto de termos verificado que esta era a melhor solução para minimizar o erro de rotação (ângulo) que o robot efectuava depois de realizadas algumas experiências, com diferentes combinações de nº de *steps*, velocidades e ângulos. Por outro lado, como já foi referido, a correcção interna que a função faz no ângulo de rotação pretendido foi obtida também à custa de experiências com vários valores de velocidades e ângulos e o valor em que fixámos a variável *erro* foi de 5 graus. Embora haja um compromisso entre o *erro* e a velocidade de rotação, este é o valor que achamos mais adequado para as velocidades mais utilizadas.

O valor do parâmetro *v* tem como resultado a imposição de velocidades simétricas nas rodas em cm/s. Considerando que V_R é a velocidade imposta nas rodas num determinado momento em cm/s, o robot estará a rodar a uma velocidade $\frac{360 * V_R * 10^{-2}}{2 * \pi * 0.31}$ graus/segundo. Em termos práticos, e segundo o manual do fabricante, para uma velocidade absoluta para cada roda de por exemplo 10 cm/s a velocidade de rotação será de aproximadamente de 20 graus/s. Como a velocidade das rodas até atingir o ângulo pretendido é composta por 3 *steps* conforme o gráfico anterior, então a velocidade efectiva da realização da acção de rotação será sempre inferior.

O mais importante de salientar é que não se devem utilizar velocidades muito elevadas ($V_R > 30$ cm/s) para este tipo de movimentos devido a provocarem erros muito grandes no ângulo de rotação e ao risco de colisões violentas com possíveis objectos ou pessoas que apareçam na trajectória do robot. Tendo em conta a natureza especifica desta acção, os valores para o pârametro velocidade aconselhados serão de 10 a 20 obtendo com estes valores um erro final práticamente desprezável.

Os ângulos de rotação possiveis são de [-180°, 180°] podendo assim ser obtida qualquer direcção. A utilização deste tipo de escala prende-se com o facto de que o sistema operativo ***Albatros™*** utilizar este tipo de escala no sistema odométrico.

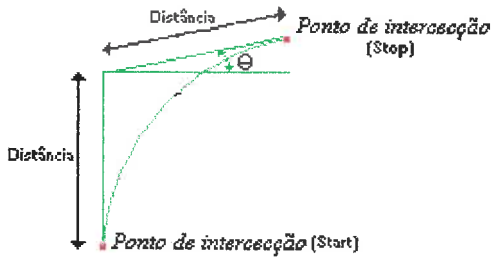
Exemplo de invocação da função *roll* :



roll(-60,20)

4.1.3. A acção *curve*

A acção *curve* implementa a concordância entre dois pontos, ou seja, efectua uma mudança de direcção através de um movimento que descreve uma circunferência, tangente a duas rectas concorrentes, em que cada ponto de convergência é equidistante do ponto de intercepcção das duas rectas. Os pontos de tangência são a posição de partida e de chegada. Assim, os parâmetros de entrada da função são: a distância (cm) dos pontos de tangência ao ponto de intercepção das rectas, o ângulo (em graus) que as rectas fazem entre si e a velocidade média (cm/s) das rodas.

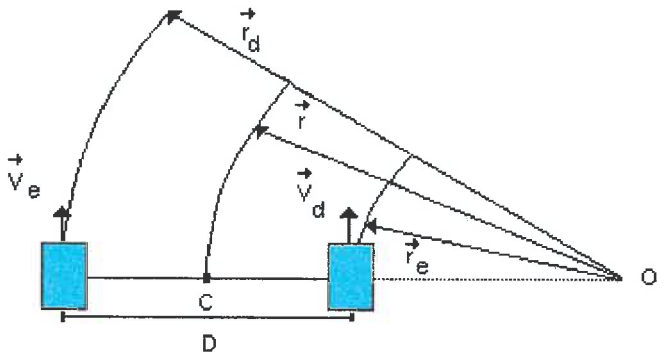


A função *curve* recorre a uma estrutura do tipo **S_CURVE** onde se guardam as variáveis internas da máquina de estados e cuja estrutura de dados definimos do seguinte modo:

```
typedef struct      {
      int state;                  /* estado da função curve  */
      long int old_x ;            /* valor inicial de x      */
      long int old_y;             /* valor inicial de y      */
      long int old_theta          /* valor inical de theta   */
      float vr;                   /* velocidade ( motor direito)    */
      float vl;                   /* velocidade ( motor esquerdo)   */
      int erro;                   /* erro associado ao movimento    */
      } S_CURVE;
```

Tal como nas funções *go* e *roll* os valores de odometria são inicializados no fim de serem guardados nas variáveis correspondentes da estrutura anterior. Para as velocidades que são impostas nas rodas são utilizadas as variáveis `vr` e `vl`, sendo mais uma vez incluida a variável `erro` que é utilizada para compensar o deslocamento excessivo provocado pela paragem.

Determinação do raio de curvatura:



Tendo em consideração a figura anterior, onde D representa a distância entre as rodas motrizes, temos que a velocidade angular do robot em relação ao ponto C é:

$$\omega = \frac{V_e}{r+\frac{D}{2}} = \frac{V_d}{r-\frac{D}{2}} \qquad \text{ou seja} \qquad \begin{cases} V_e = \omega r + \frac{\omega D}{2} \\ V_d = \omega r - \frac{\omega D}{2} \end{cases}$$

A velocidade linear e a velocidade angular do ponto C poder-se-ão escrever respectivamente como:

$$V = \frac{V_d + V_e}{2} \qquad \text{e} \quad \omega = \frac{V_e - V_d}{D}$$

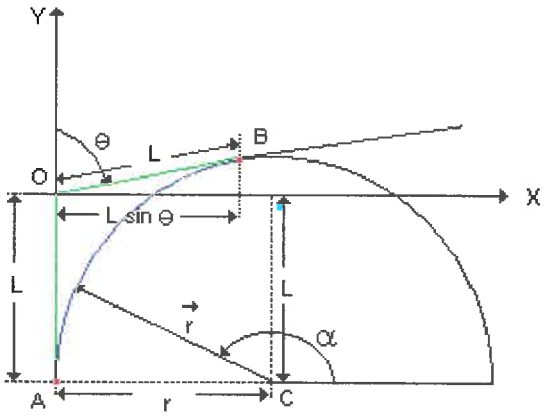
Como $r = \frac{V}{\omega}$, vem que $r = \frac{V_d + V_e}{V_d - V_e}\frac{D}{2}$

Podemos determinar velocidades para as rodas esquerda e direita para que o robot possa efectuar uma determinada circunferência conhecido o seu raio.

Isto é, tendo em conta as expressões anteriores podemos determinar que:

$$V_d = 2V - \frac{V(r-\frac{D}{2})}{r} \qquad ; \qquad V_e = \frac{V(r-\frac{D}{2})}{r}$$

É importante salientar que teremos todo o interesse em determinar r, conhecidos o ângulo de curvatura θ, a distância em que o robot deverá curvar L, e a sua velocidade (linear) V. Assim de acordo com a seguinte figura o robot deverá efectuar a trajectória do ponto A para o ponto B (representada a azul na figura) conhecida a velocidade linear, a distância L (a verde) e o ângulo de curvatura θ.



Começaremos por considerar que no sistema representado com origem no ponto O:

- θ é o ângulo formado pela direcção OY e o segmento de recta OB.
- O ponto B tem coordenadas (Lsinθ, Lcosθ).
- A curva a azul é um troço da circunferência com raio r (concordância) e centro no ponto C = (r, -L).

Então considerando as equações paramétricas de uma circunferência de centro no ponto C e que passa pelo ponto B, temos que:

$$\begin{cases} L\sin\theta = r + r\cos\alpha \\ L\cos\theta = -L + r\sin\alpha \end{cases} \Leftrightarrow \begin{cases} r = \dfrac{l(\cos(\theta)+1)}{\sin(180-\theta)} \\ r = \dfrac{l\cos\theta + l}{\sin\alpha} \end{cases}$$

ou seja:

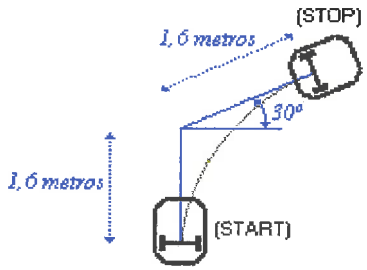
$$r = \frac{l(\cos(\theta)+1}{\sin(180°-\theta)} \quad ; \qquad \text{Considerando que: } \sin(180°-\theta) = \sin(\theta)$$

$$r = \frac{L(\cos(\theta)+1)}{\sin(\theta)}$$

Temos finalmente o valor de r em função da distância L e do ângulo θ, com o qual determinaremos as velocidades para as rodas esquerda e direita, com as quais o robot efectuará a curvatura pretendida.

Tal como como na função anterior são feitos *steps* de velocidade intermédios para minimizar erros associados à paragem. Estes *steps* são controlados através da análise do ângulo de curvatura que o robot faz em relação à posição inicial.

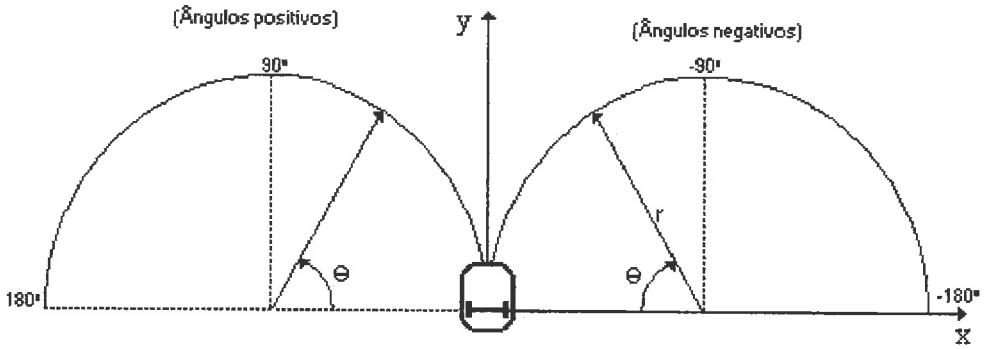
Exemplo de invocação:



curve(160,-30,20)

4.1.4. A acção *circ*

A função ***circ*** executa um movimento que descreve um arco de circunferência com raio r (cm) durante um ângulo θ (graus) a uma velocidade média de v (cm/s). Assim, os parâmetros de entrada são respectivamente: o raio r, o ângulo θ e a velocidade v. Obviamente esta função é uma derivação da função anterior (***curve***) sendo neste caso o raio um parâmetro de entrada.

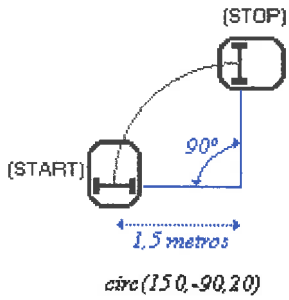


Tal como as funções anteriores, tem uma estrutura associada do tipo **S_CIRC** que contêm as variáveis internas da função.

```
typedef struct      {
      int state;                  /* estado da função circ          */
      long int old_x;             /* valor inicial de x             */
      long int old_y;             /* valor inicial de y             */
      long int old_theta;         /* valor inical de theta          */
      float vr;                   /* velocidade ( motor direito)    */
      float vl;                   /* velocidade ( motor esquerdo)   */
      int erro;                   /* erro associado ao movimento    */
      } S_CIRC;
```

A estrutura, bem como o algoritmo desta função é semelhante à utilizada pela função ***curve***, a necessidade da existência desta função, existindo já a função anterior, prende-se com o facto de que para o utilizador, em certas situações, a utilização da função ***circ*** é obvia enquanto que com a função ***curve***, isso já não acontece.

Exemplo de invocação:



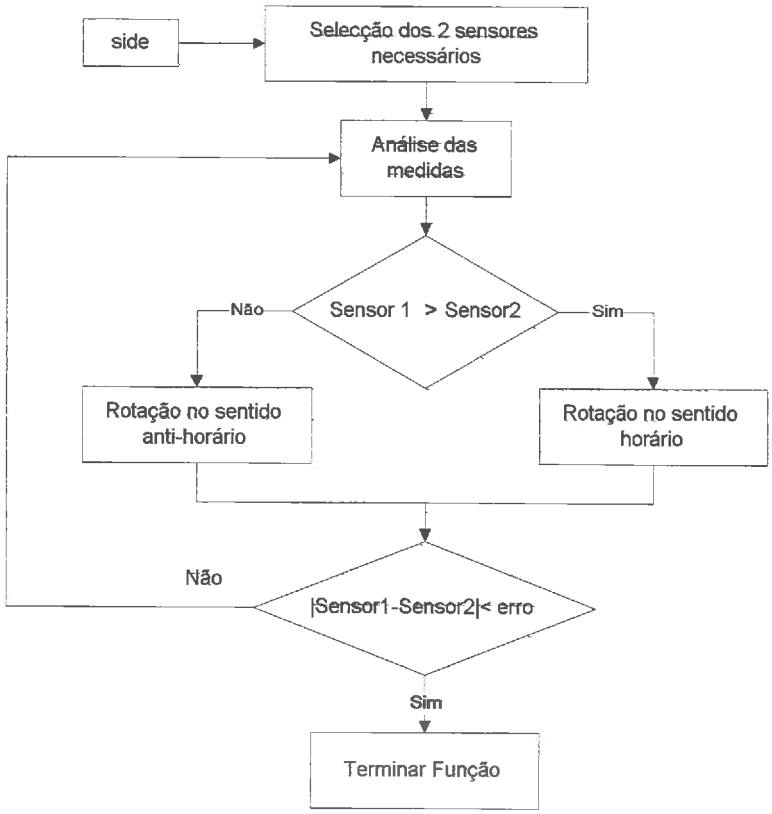
4.1.5. A acção *parallel*

Esta função executa um movimento de rotação com uma velocidade mínima até que a orientação do robot fique paralela ao lado escolhido, e que é o parâmetro de entrada (`FRONT`, `RIGHT`, `BACK` e `LEFT`). Esta manobra do tipo 2 (ver secção 4.1) é util para ajustamentos da trajectória nas zonas de calibração facilitando assim a construção da missão.

A função *parallel* recorre a uma estrutura do tipo **S_PARALLEL** onde se guardam as variáveis internas da máquina de estados e cuja estrutura de dados definimos do seguinte modo:

```
typedef struct      {
      int state;                  /* estado da função parallel      */
      int erro;                   /* erro associado ao movimento    */
      } S_PARALLEL;
```

O algoritmo desta função está esquematizado na figura seguinte:

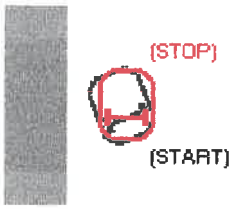


Como podemos verificar neste caso a variável *erro* tem uma função diferente das variáveis com o mesmo nome das funções anteriores. Aqui, o *erro* é a diferença de valores medidos nos sensores seleccionados, podendo assim configurar a precisão na obtenção do paralelismo pretendido. O valor é o mais pequeno possível embora quanto mais pequeno for,mais tempo o robot necessitará para executar esta acção (idealmente os valores seriam iguais). O valor que achamos razoavel nos testes é 2, tendo em conta também a precisão que temos nos sensores de ultra-som.

Os únicos cuidados a ter na utilização desta função são, por um lado não fazer calibrações com distâncias da parede correspondente muito grandes (valor óptimo será entre 50 a 80 cm) e escolher zonas onde não seja facilmente alterável a configuração da superficie plana.

A primeira advertência vai para as grandes distâncias, onde a precisão das medidas diminui tornando o erro demasiado elevado para obter resultados aceitáveis. A segunda advertência prende-se com o facto de que embora na altura da construção da missão a zona de calibração seja óptima (não deformável, não conter partes móveis, pouco frequentada por pessoas ou objectos, etc...), se não se tiverem certos cuidados, numa futura execução da missão essa mesma zona pode estar alterada (deslocada ou até simplesmente ter deixado de existir).

Exemplo de invocação:



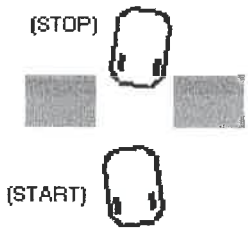
parallel(LEFT)

4.1.6. A acção *cross_door*

Esta função do tipo 3, activa o módulo *doors* desenvolvido no trabalho anterior, [Ranger98] e que é responsável pelo atravessamento de passagens estreitas, efectuando todo o controlo sobre o robot até que este tenha concluido a operação. Esta acção só deve ser executada quando o robot se encontra aproximadamente em frente de uma passagem estreita. No fim do atravessamento o módulo activa a *flag* ***success***, sendo o controlo do robot novamente retomado pela função ***cross_door***. Esta função termina desactivando o módulo *doors* e inicializando a variável ***success***.

Esta acção tem alguns problemas de estabilidade: por vezes o módulo faz uma abordagem errada à passagem estreita. Este problema poderá ser resolvido com um refinamento do módulo.

Exemplo de invocação:

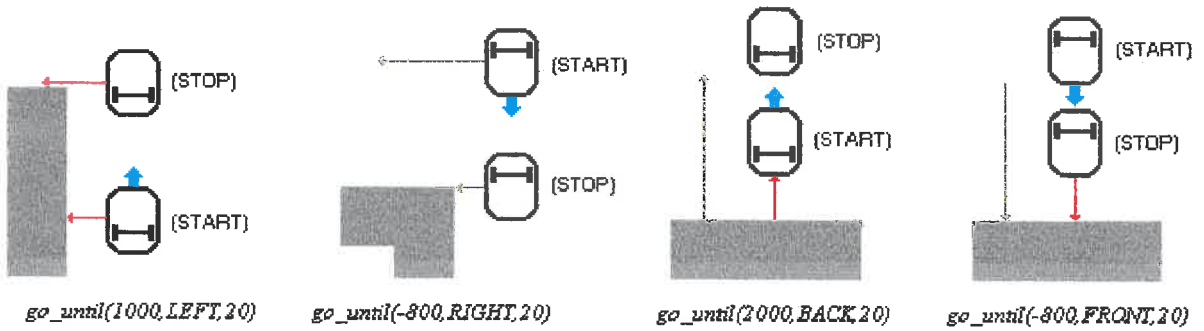


cross_door()

4.1.7. A acção *go_until*

Esta acção executa um movimento rectilíneo para a frente ou para trás, até que aconteça uma dada condição, através da leitura dos sensores de ultra-sons. Essa condição é construida através dos valores dos parâmetros de entrada que são: distância (cm), lado(`FRONT`, `RIGHT`, `BACK` e `LEFT`). O terceiro parâmetro é velocidade(cm/s) a que o robot se vai deslocar. O parâmetro lado, indica ao robot onde vai ser testada a condição. A condição é contruida indicando a distância de fronteira entre a medição ser maior ou menor, e através disso, verificar se foi atingido o objectivo. Nas figuras seguintes indicamos algumas possibilidades de aplicação.

Exemplo de invocação:



go_until(1000,LEFT,20) *go_until(-800,RIGHT,20)* *go_until(2000,BACK,20)* *go_until(-800,FRONT,20)*

A utilização desta acção requer alguns cuidados, pois a existência de um objecto ou a passagem de uma pessoa na zona de medida da distância, que serve para formar a condição de paragem, faz com que, momentâneamente, a distância medida seja alterada para um valor, que do ponto de vista da missão, leva a que a função possa tome a decisão errada. Este tipo de erro só acontece se o valor da distância referida for negativo.

4.1.8. A acção *go_parallel*

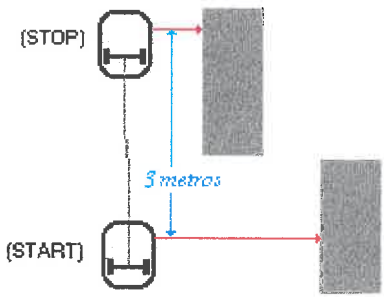
Esta acção executa um movimento semelhante ao da acção ***go***, mas com o acréscimo de fazer a correcção contínua da trajectória, na tentativa de manter o robot o mais paralelo possível ao lado indicado (2º parâmetro, RIGHT ou LEFT). O primeiro e o terceiro parâmetros são a distância (cm) e a velocidade (cm/s).

A função *go_parallel* recorre a uma estrutura do tipo **S_GO_PARALLEL** onde se guardam as variáveis internas da máquina de estados e cuja estrutura de dados definimos do seguinte modo:

```
typedef struct {
      int state;                    /* estado da função go_parallel   */
      long int old_x;               /* valor inicial de x             */
      long int old_y;               /* valor inicial de y             */
      long int old_theta;           /* valor inicial de theta         */
      int vr;                       /* velocidade (roda direita)      */
      int vl;                       /* velocidade (roda esquerda)     */
      int erro;                     /* erro associado ao movimento    */
      } S_GO_PARALLEL;
```

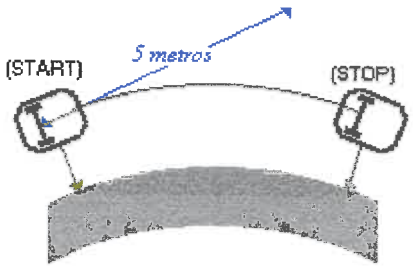
A descontinuidade das supefícies sobre as quais estão a ser medidas as distâncias, não afecta o desempenho do algoritmo, sendo esta a principal vantagem da função.

Assim, como mostra a figura seguinte, que ilustra um exemplo de aplicação, uma diferença de por exemplo 2 metros entre duas superficies (medida pelo robot), vai afectar a trajectória como se a diferença fosse 2 cm, e como o tempo de atravessamento da descontinuidade é reduzido, o ajuste de trajectória é pouco significativo, recuperando o paralelismo.



go_parallel(300,20,RIGHT)

Outra situação possível de utilização desta accção é o seguimento de superficies curvas quando a geometria precisa das curvaturas não é bem definida, e pouco pronunciada. A figura seguinte mostra um exemplo de uma aplicação que descreve uma situação semelhante.



go_parallel(500,20,RIGHT)

Na situação da figura anterior, a distância (1º parâmetro de entrada) ao fim da qual o robot se imobiliza, é o comprimento medido pelo sistema odométrico no eixo das abcissas, com a origem dos eixos no ponto de partida.

As curvas possiveis para esta acção podem ser mais elaboradas, não podendo no entanto, as ter curvaturas muito pronunciadas pois assim, o robot não teria tempo de efectuar uma correcção efeciente. Como a correcção é obtida através da soma ou subtração de uma quantidade fixa, às velocidades das rodas, os valores utilizados para parâmetros de entrada não poderão ser muito elevados, já que iriam compremeter a eficiência da acção.

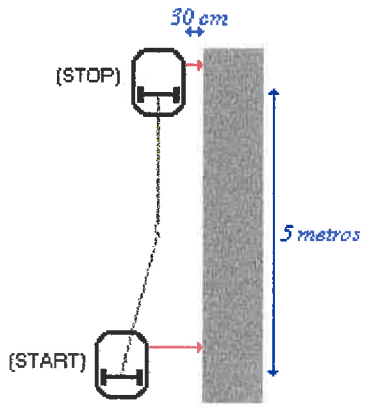
4.1.9. A acção *go_at*

Esta acção, tal como a anterior (*go_parallel*), executa um movimento semelhante à acção ***go***, ou seja, percorre uma determinada distância (cm), que é o primeiro parâmetro de entrada, mas com o acréscimo de fazer a correcção contínua da trajectória, na tentativa de manter o robot o mais paralelo possivel ao lado indicado (3º parâmetro, RIGHT ou LEFT), e a uma dada distância (cm) desse mesmo lado (2º parâmetro). Assim, é imposta uma correcção continua nas velocidades das rodas, e quando é obtida a distância lateral pretendida, medida pelos sensores nesse lado, tenta continuamente mantê-la. A acção termina, com a imoblização do robot, quando já foi percorrida a distância indicada pelo primeiro parâmetro, medida pela odometria, da mesma forma que acção *go_parallel*.

De salientar que, se o robot não estiver relativamente próximo(<100 cm) da distância lateral pretendida, a correcção que é imposta não vai conseguir compensar, dentro do espaço a percorrer, a trajectória de modo a atingir a proximidade lateral pretendida, além do facto de que, quanto maior for a velocidade imposta (4º parâmetro), menor peso terá a correcção nos valores das velocidades dos motores.

Uma solução para este facto seria existir um erro proporcional à velocidade de entrada e à diferença entre a distância (medida pelos sensores) e a pretendida. Com uma solução deste tipo, existe outro problema que é o de uma correcção excessiva poder resultar numa curvatura tal, que o robot ficaria de frente para a superficie que anteriormente era lateral, o que não é o intuito desta acção que, como foi dito no inicio, é uma acção derivada da acção *go*.

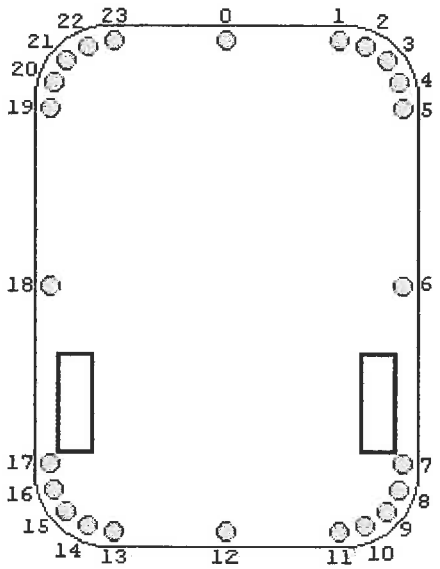
Na figura seguinte está representado um exemplo de aplicação:



go_at(500, 30, RIGHT, 20)

4.1.10. A acção *active_us*

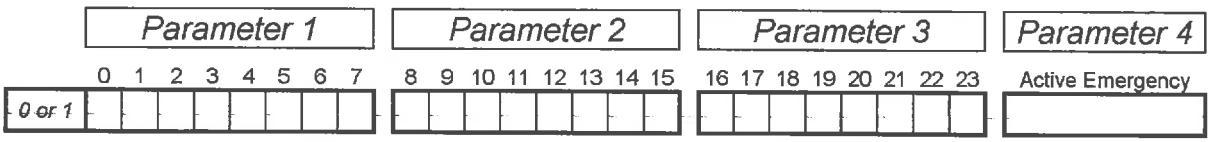
Esta acção configura quais os sensores que vão ficar activos depois da sua execução, activando também, ou desactivando a detecção de emergência através dos sensores de ultra-sons.



Posição no robot dos sensores de ultra-sons (vista de cima)

Esta função é útil para situações de aproximação muito críticas, ou para casos onde a interferência do ambiente que rodeia o robot numa determinada zona, possa prejudicar os objectivos de uma missão, garantindo sempre o facto de, não pode haver colisões com pessoas ou objectos.

O quadro seguinte representa o modo de configurar o estado dos sensores através da colocação a 0 ou 1 no campo correspondente.



Exemplo:

active_us(10000010,00010000,00100000,0);

O exemplo anterior activa os 4 sensores centrais (0, 6, 12 e 18) de cada lado do robot, ficando os restantes desactivados. Além disso a detecção de emergência através de sensores de ultra-sons é também desactivada.

4.2. Módulo *serial.c (gestor de missões)*

Tal como na versão anterior, este módulo é responsável pela comunicação com o exterior através da porta série. Além das três categorias já implementadas foram acrescentadas duas novas categorias que estão relacionadas com o gestor de missões que foi implementado. Assim, existe um protocolo com 5 categorias de mensagens:

Tipo A – Envio de comandos;
Tipo B – Pedido de informação ao robot;
Tipo U – Mensagens associadas aos ultra-sons.
Tipo **T** – *Download* de acções que formam uma missão.
Tipo **C** – Comandos isolados para gestão de missões.

Para além de testar se existem mensagens, este módulo trata-as, seja através do envio da respectiva resposta ou do processamento dos comandos que essas mensagens possam constituir.

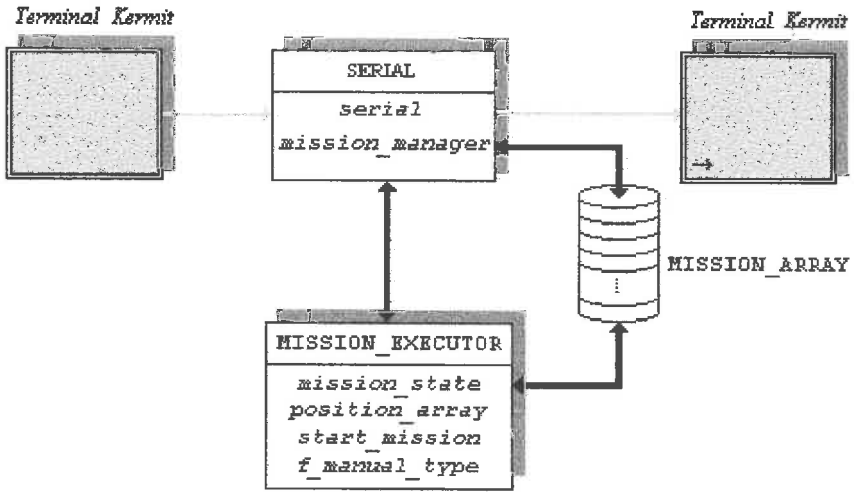
O formato utilizado para as mensagens é baseado sobre pesquisas anteriores desenvolvidas na área de comunicação com plataformas móveis.

| category | | Type | | Parameter 1 | | Parameter 2 | .....

Com o formato estabelecido, foi construido um sistema que permite ao utilizador fazer a gestão de uma missão directamente no robot. Esta gestão engloba o carregamento de um conjunto de acções, a sua manipulação através de comandos de remoção, inserção ou alteração isolada, o comando de começo, pausa ou fim da execução da missão ou mesmo remoção desta através da inicialização do array de acções. Por outro lado, dotamos o sistema de comandos que executam uma acção isolada, sem que estes interfiram com a gestão da missão, ou seja, que se tenha que fazer alguma alteração no array de acções. Esta funcionalidade é util para efectuar manobras isoladas com o robot, ajustes ou testes de acções que possam facilitar o planeamento de uma missão. Um exemplo dessa situação é se na construção de uma missão estiverem a serem feitos os primeiros testes, e uma manobra foi mal implementada, basta ao utilizador fazer uma pausa da execução da missão, corrigir a posição do robot através de uma acção isolada e retomar a execução sequencial da missão. Assim, não é necessário carregar novamente a missão e esperar que o resultado da alteração seja executado.

Toda a gestão de uma missão é feita sobre um *array* de acções que está ligado ao módulo ***serial.c***, sendo a execução da missão e dos comandos isolados, da responsabilidade do módulo ***mission_executor.c***.

Na figura seguinte, estão representadas as ligações importantes, do ponto de vista da gestão de missões, entre os módulos que intervêm no envio e execução de uma acção.



A categoria T é utilizada para o preencimento do array de missões, ficando armazenadas nesse *array* todas as acções que sejam enviadas pela porta série, através do terminal *kermit*. À medida que chegam, as acções enviadas por esta categoria de mensagem, são ligadas às restantes, formando a lista ligada de acções que formam a missão. Uma maneira de construir uma missão e preencher o *array* de acções de uma forma rápida, e que pode ser reutilizada no futuro, é o de escrever um ficheiro de texto, com as várias acções, uma em cada linha, constituindo cada linha uma mensagem do tipo T. De seguida, faz-se o *download* do ficheiro para a porta série, sendo descodificada cada mensagem e efectuado o devido preenchimento do array de acções. A codificação das acções em parâmetros da categoria T, é indicada mais à frente na descrição do modo manual de execução de uma acção, o qual utiliza o mesmo protocolo de codificação.

Exemplo de algumas mensagens desta categoria:

```
.....
T 1 100 20
T 2 90 15
T 1 100 20
T 2 90 15
.....
```

A categoria C, para execução de comandos isolados de gestão da missão, contém vários tipos de operações os quais se encontram ilustrados na tabela seguinte:

C Category (*isolated commands for mission management*)		
Type	*Description*	*Example*
A	Start the mission processing	CA
S	Stop the mission processing	CS
P	Pause	CP
E	Clear the task mission array	CE
D	Delete a task in mission array	CD 25
R	Replace an exintent task	CR 12
I	Insert a task in mission array	CI 10 3 100 20 90
L	Give the list of tasks to execute	CL
T	Manual tasking processing	CT 4 100 90 20

De um modo geral, podemos dividir os comandos desta categoria em dois grupos: os de gestão propriamente dita (E, D, R, I e L), e os de controlo de execução (A, S, P e T).

- **Comandos de gestão**:

CE → Faz a inicialização do array de acções, interrompendo qualquer função que esteja a ser executada e imoblizando o robot, se necessário.

CD → Apaga o registo de uma acção numa determinada posição do array, inicializando essa posição que fica livre para futura utilização. O parâmero de entrada é o número de referência dessa mesma posição no array de acções.

CR → Efectua a reprogramação dos vários campos, de uma determinada posição do array de acções, podendo ser assim alterada uma acção que, por algum motivo não tenha sido devidamente construida. O primeiro parâmetro de entrada é a referência da posição no *array* onde vai ser efectuada a operação, sendo os parâmetros seguintes respeitantes à programação da acção.

CI → Insere uma posição na lista de execução de tarefas a executar, sendo o primeiro parâmetro a referência da posição (no array de missões) que precede a nova acção, sendo os restantes parâmetros respeitantes à nova acção. Assim, se por exemplo quisermos inserir uma acção, a seguir à n-ésima acção a ser executada, faz-se a listagem da missão e com o número de referência da n-ésima missão, no primeiro parâmetro do comando de inserção, preenchemos os restantes parâmetros com a nova acção.

CL → Faz a listagem de todas as acções prontas a executar, que o *array* de missões contém, fazendo a sua numeração automáticamente, por ordem de chegada. Assim, a lista contém o número de execução, o número de referência e a tradução em linguagem C da acção.

- **Comandos de controlo:**

CA → Inicia a execução da missão, ou seja, todas as acções que o *array* contém, são executadas sequencialmente, até a paragem de execução. Este comando não pode ser executado se estiver a ser processada uma tarefa em modo manual, ou seja, através do comando CT.

CS →Paragem da execução da missão, sendo feito o *reset* à acção que supostamente estaria a ser executada. Assim, se a execução da missão for reiniciada, a execução da primeira acção será a seguinte à que foi interrompida, na altura da paragem.

CP → Pausa na execução da missão, sendo efectuada a imobilização do robot. Se for novamente executado o comando, será retomada a execução da missão no ponto onde foi interrompida.

CT → Execução de uma tarefa isoladamente, tendo que terminar a execução desta para que possa ser executada um novo comando deste tipo. Este comando não pode ser executado se a missão estiver em execução. Refira-se que este tipo de execução de tarefas não interfere com o *array* de acções, podendo, como já foi,mencionado anteriormente, ser executado no meio de uma missão, desde que essa missão esteja interrompida. O modo como se preenchem os vários parâmetros que vão indicar ao executor de acções qual é a acção, e os respectivos parâmetros, está na tabela seguinte:

Protocol to edit a task (manual or in download mode)					
Task name	*type*	*Parameter 1*	*Parameter 2*	*Parameter 3*	*Parameter 4*
go	1	*distance (cm)*	*velocity (cm/s)*	-	-
roll	2	*ângulo (º)*	*velocity*	-	-
curve	3	*distance (cm)*	*ângulo*	*velocity (cm/s)*	-
circ	4	*ray (cm)*	*ângulo (º)*	*velocity (cm/s)*	-
parallel	5	*Side*	-	-	-
go_until	6	*distance (cm)*	*Side*	*velocity (cm/s)*	-
cross_door	7	-	-	-	-
go_parallel	8	*distance (cm)*	*Side*	*velocity (cm/s)*	-
go_at	9	*distance (cm)*	*lateral distance (cm)*	*Side*	*velocity (cm/s)*
active_us	10	*sensor 0 to 7*	*sensor 8 to 15*	*sensor 16 to 23*	*emergency(ON/OFF)*
.....	..	.....	.....	.....	.....

Como se pode ver pela tabela anterior, cada acção é identificada por um número, que constitui sempre o primeiro parâmetro destinado à acção, os restantes parâmetros são os campos respectivos, necessários para a sua execução.

Exemplo de uma execução da acção ***circ*** em modo manual:

```
CT 4 100 90 20
```

Para que o gestor de missões possa interpretar novos tipos de acções , apenas terá que ser introduzido na categoria correspondente o novo tipo de acção, sendo o processo de descodificação dos parâmetros semelhante aos já existentes.

4.3. Módulo *mission_executor.c*

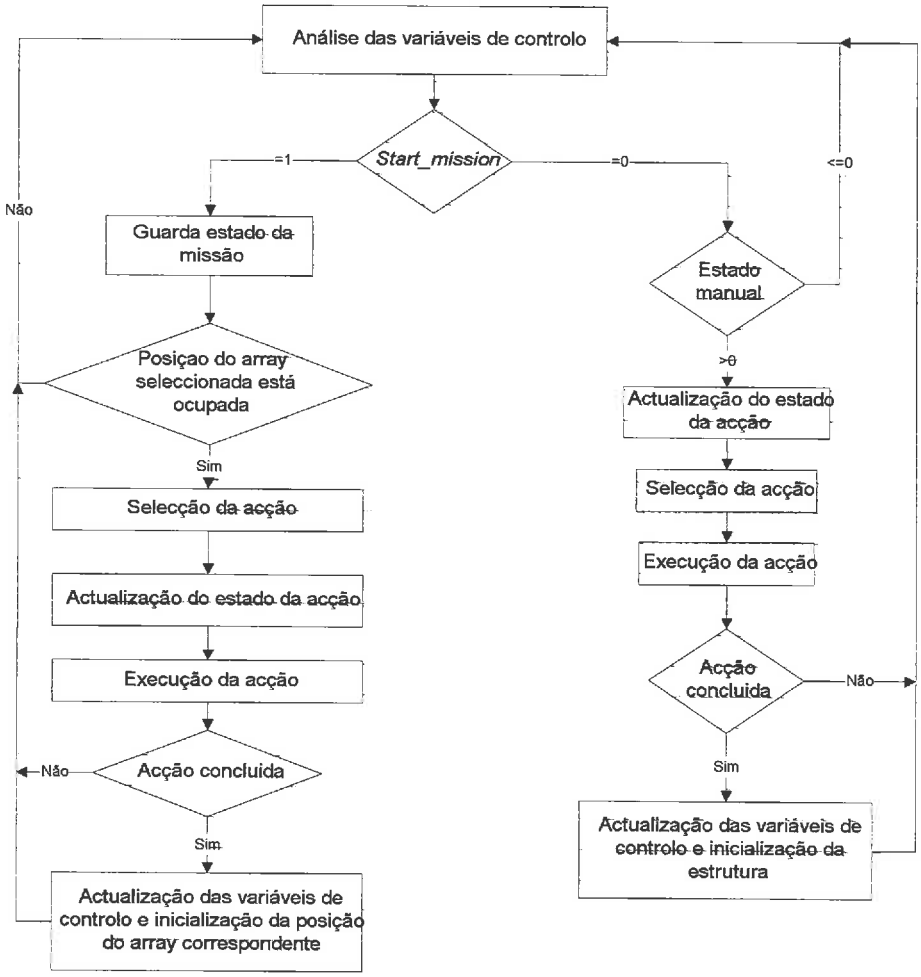
Este módulo é o responsável pela execução das acções, ou seja, é neste módulo que se encontra o algoritmo de invocção de cada acção. Por outro lado, é capaz de intrepretar os parâmetros que estão armazenados no *array* de acções, seleccionar qual posição do *array* correcta, e implementar a execução efectiva da acção.

Existem neste módulo, quatro variáveis globais (*mission_state*, *position_array*, *start_mission* e *f_manual_type*) que fazem a comunicação entre este módulo e o gestor de missões pertencente ao módulo *serial.c*.

A variável *f_manual_type* recorre a uma estrutura do tipo S_MANUAL_TYPE onde são guardados os vários componentes de uma acção, para a sua execução em modo manual, não recorrendo ao *array* de acções.

```
typedef struct {
      int state;          /* Estado da acção (em execução ou não)              */
      int mission_state;  /* Estado da missão antes da execução em modo manual */
      int type;           /* Tipo da acção a executar                          */
      int parameter1;     /* Primeiro parâmetro                                */
      int parameter2;     /* Segundo parâmetro                                 */
      int parameter3;     /* Terceiro parâmetro                                */
      int parameter4;       /* Quarto parâmetro                                */
      } S_MANUAL_TYPE;
```

O algoritmo para a execução de uma acção, pode ser representado pelo seguinte diagrama:



Como se pode ver pelo diagrama anterior, o modo manual de execução de uma acção é totalmente independente do modo automático. Embora o módulo tenha a parte de invocação de acções distinta, as rotinas de cada acção são as mesmas. Se houver a necessidade de introduzir novos tipos de acções, as novas rotinas teram apenas que, no seu algoritmo de controlo, quando terminadas, incrementar a variável *mission_state* para que o algoritmo de controlo, detecte o fim da acção. Por outro lado, terá que ser introduzido nos dois modos, a sua forma de invocação com os devidos parâmetros. Se for necessária uma alteração de uma função, ao nível do algoritmo interno, o gestor de missões é completamente transparente a essa transformação.

A interrupção de uma acção por parte de outro módulo directamente sobre este, é feita alterarando o estado da acção que está em execução. Os algoritmos de cada acção, têm todos uma estrutura semelhante ao nível das máquinas de estados que os integram, sendo compostos por uma inicialização, execução e teste de fim de execução. Esta situação pode ser vista no contexto de que se houver um módulo, que integre este sistema, e que inclua nos seus algoritmos internos detecção

e tomada de decisões, como é o caso da detecção de *landmarks* distribuidas (ex: contagem de portas), este pode actuar no fluxo da missão, podendo interagir com o gestor de missões para configuração dos parâmetros internos ao sua execução.

4.4. Utilização das ferramentas desenvolvidas

Com o objectivo de utilizar as funções que iam sendo desenvolvidas, foram sendo preparadas pequenas demonstrações (missões) as quais serviam para testar e refinar as acções entretanto construídas. As principais demonstrações, antes da construção do gestor de missões são:

- *exagono_curve_45*;
- *exagono_curve_60*;
- *demo_abril1*;
- *demo_abril2*;

Destas demonstrações destacamos a *demo_abril1*, *demo_abril2* e a *openday*, que consistia em sair do Laboratório de Automação e Robótica, ir até ao elevador de serviço e voltar a entrar no laboratório, posicionando-se no local de partida.

Quando foi possivel a edição através do gestor de missões, as demonstrações foram várias, devido ao facto de se poderem facilmente construir, servindo para testes e *debug* do próprio gestor de missões. Mesmo assim, destacamos a última (*pórtico*), que foi construida, para exemplificação, que consistia em, partindo do local pré-defenido, dentro do Laboratório de Automação e Robótica, deslocar-se até ao Laboratório de Célula Flexivel de Produção, e estacionar-se por baixo do portico existente.

4.4.1. Demonstrações

A demonstração *demo_abril1*, foi efectuada com conhecimento prévio da sala e sem grande precisão relativamente a medidas efectuadas. Conhecíamos a sala e sabíamos por exemplo que o robot podia andar para a frente mais ou menos x metros ou que para fazer uma determinada mudança de direção, esta poderia ser feita com segurança em mais ou menos 1 metro (por exemplo), ou ainda que se o robot fosse a uma dada velocidade, não se deveria aproximar de um obstáculo menos de x metros para que a sua segurança não fosse posta em causa.

Assim para construir uma missão temos duas fases:

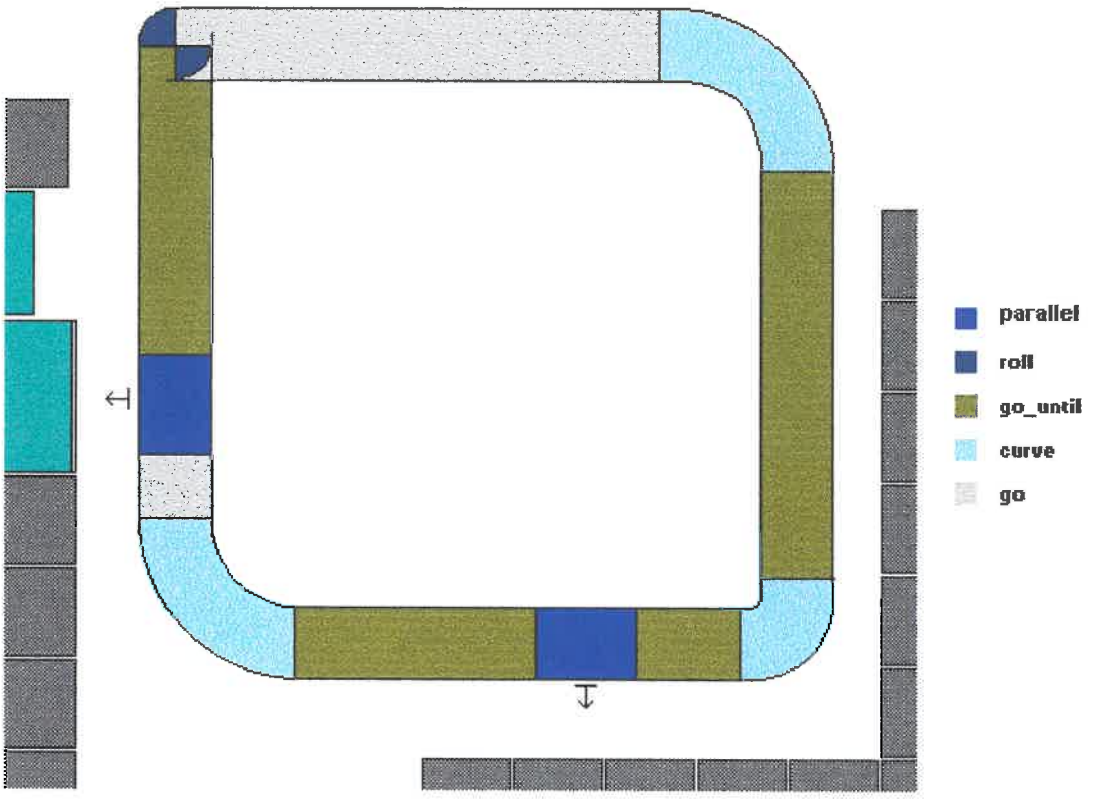
1 - Fazer o plano da missão em papel com medidas mais ou menos precisas dos movimentos necessários.

2 – Intregrá-la no código.

É importante referir que por exemplo para a demonstração demo_abril1 foram gastos aproximadamente 10 minutos para a 1ª fase mais 5 minutos para que fosse efectuado o 1º teste com o robot (para os programadores).

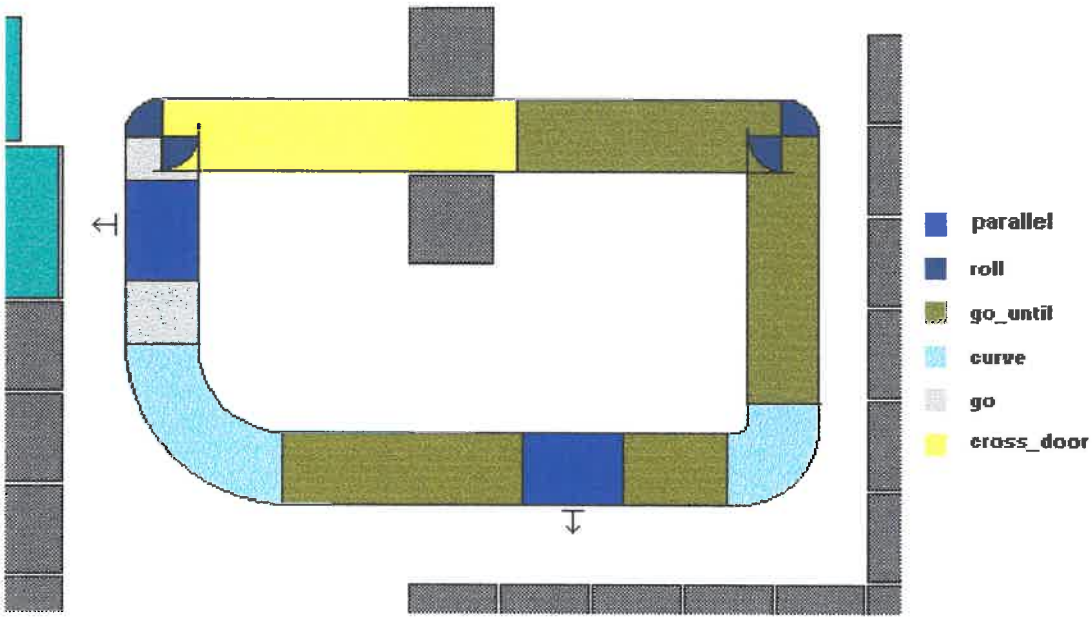
As figuras seguintes pretendem ser uma representação simples das missões *demo_abril1*, *demo_abril2*.

demo_ abril1



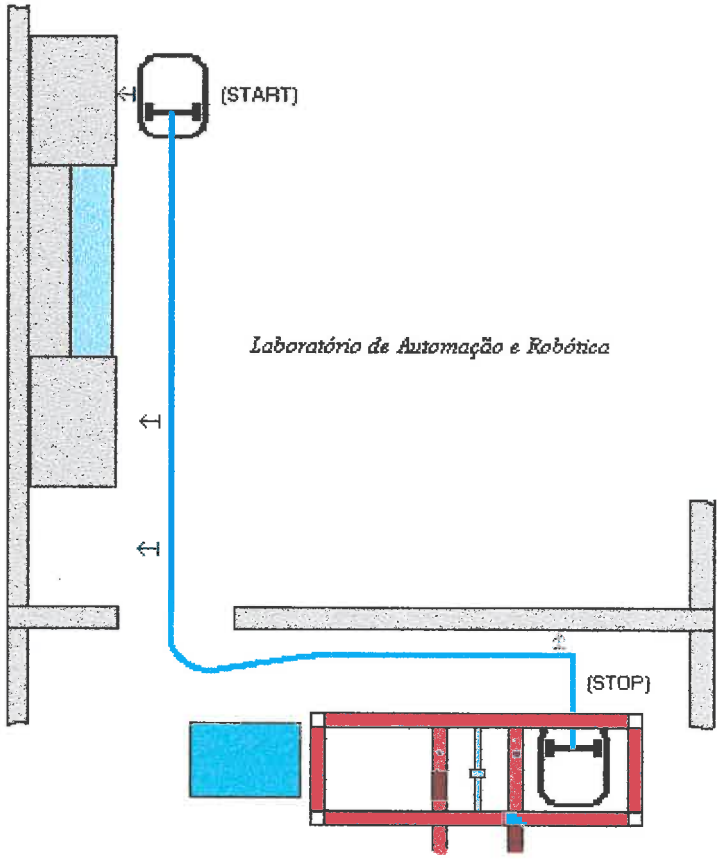

```
parallel(LEFT);
go_until(-150,FRONT,20);
curve(110,-90,20);
go(50,20);
parallel(LEFT);
go_until(200,LEFT,20);
go(80,20);
roll(-90,10);
go(150,20);
curve(200,-90,20);
go_until(-170,FRONT,20);
curve(10000,-90,20);
go_until(200,BACK,20);
```

demo_ abril2




```
parallel(LEFT);
go_until(-160,FRONT,40);
curve(110,-90,40);
go(30,20);
parallel(LEFT);
go(100,20);
roll(-90,10);
cross_door();
go_until(-80,FRONT,30);
roll(-90,20);
parallel(LEFT);
go_until(-170,FRONT,40);
curve(100,-90,40);
go_until(20,BACK,40);
```

pórtico



Código respeitante à demonstração de deslocamento ao pórtico pertencente à célula flexivel de produção. O conteúdo do ficheiro de texto para *download* da missão é o seguinte:

```
T 5 4
T 2 -180 10
T 9 250 30 2 15
T 9 100 25 2 15
T 5 2
T 6 80 2 10
T 1 70 10
T 5 2
T 1 150 10
T 10 11111111 11111111 11111111 0
T 4 50 90 10
T 10 11111111 11111111 11111111 1
T 9 370 30 4 10
T 5 4
T 2 -90 10
T 1 30 10
T 9 40 4 20 10
```

5. Trabalho futuro

5.1. Aumento da flexibilidade das acções

Futuramente, algumas das acções desenvolvidas poderão vir a ser refinadas ou até acrescentadas outras, com graus de flexibilidade diferentes dos utilizados. Referimos por exemplo que algumas acções poderão assumir parâmetros por defeito, caso o utilizador não os introduza (ex: velocidade), podendo noutras, eliminar parâmetros.

Poderá ainda ser desenvolvida uma acção de contorno de obstáculos inesperados, de certa forma semelhante à acção *go_at*, mas na qual a velocidade,ou será calculada e imposta nas rodas de acordo com a situação, ou poderá, simplesmente ser um valor assumido por defeito.

5.2. Módulo *detect*

O módulo *detect.c* poderá ser considerado trabalho futuro já que se encontra numa fase inicial. Embora já estruturado ainda não foi testado não sabendo portanto se esta será a melhor solução para o pretendido.

Integrado no sistema do qual já foi feita uma descrição (módulos desenvolvidos no trabalho anterior) poderá ser construído o módulo *detect.c* do qual descreveremos as funcionalidades que pensamos possam ser implementadas, salientando o facto de que este módulo está a ser inteiramente desenvolvido não sendo até a data, feitos quaiquer testes práticos.

Este módulo vai funcionar independentemente de todos os outros disponibilizando variáveis (globais) que indicarão por exemplo (ponto seguinte) o número de portas que o robot identificou lateralmente, supondo que robot se encontra em movimento. Estas variáveis poderão ser completamente manipuladas pelos outros módulos.

A filosofia a seguir, será de que este tipo de funções estará sempre a funcionar em continua análise do sistema sensorial e odométrico, actualizando as variáveis respectivas se necessário. Sendo inicializadas num determinado momento pelo gestor de missões (ou outro qualquer), tomam a partir desse momento um significado de contagem incremental das respectivas variáveis.

5.2.1. Estrutura HISTORY

Esta estrutura prentende implementar um pequena base de dados dos últimos valores obtidos (SIZE_HIST) nos sensores de ultra-sons e também os valores de odometria no instante da aquisição destes. O preenchimento da tabela estará relacionado com o movimento do robot sendo para isso convencionados *steps* (ou intervalos) para o efeito (não terá lógica estar a preencher a base de dados com o robot parado pois os valores seriam semelhantes). Assim, como mostra a figura seguinte, uma linha da base de dados será do tipo:

Como se pode ver na figura anterior definimos três classes de dados, os valores dos 24 sensores de ultra-sons, os três parâmetros de odometria e em último lugar o tipo de acção que está

a ser executada no momento do preenchimento da linha da base de dados. Para este tipo de parâmetro é indicado o respectivo valor segundo a tabela seguinte:

Nome da acção	tipo
go	1
roll	2
curve	3
circ	4
parallel	5
go_until	6
cross_door	7
go_parallel	8
go_at	9
active_us	10
.....	..

O parâmetro tipo foi incluido para o cálculo dos valores odométricos absolutos. Dado que algumas acções fazem o *reset* das variáveis odométricas, se estes valores fossem retirados directamente da variável *pos* (que o módulo *executor.c* disponibiliza) seriam inconsistentes, tanto para o cálculo dos intervalos de amostragem como para o preencimento da base de dados. Assim, quando acontecem transições entre acções, só ao fim de terminar cada acção é que os valores odométricos absolutos são repostos.

O módulo *detect.c* será responsável pela actualização permanente e autónoma desta pequena base de dados, podendo ser analizada em paralelo por varias funções como a contagem de portas que descrevemos no ponto seguinte.

5.2.2. Contagem de portas

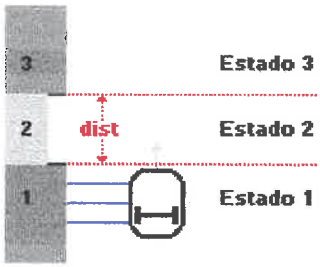
O algoritmo de contagem de portas poderá ser baseado na análise da base de dados ***hist*** com as últimas medidas dos sensores de ultra-sons sendo o variável ***n_doors*** (do tipo **S_N_DOORS**) actualizada incrementalmente à medida que são detectados padrões que identifiquem portas.

A estrutura **S_N_DOORS** descrita a seguir correponderá à contagem de portas relativamente aos lados esquerdo e direito do robot.

```
typedef struct      {
      int state_left;          /* estado de identificação(left)   */
      int left ;               /* valor da contagem (left)        */
      int state_right;         /* estado de identificação (right) */
      int right;               /* valor da contagem (right)       */
      } S_N_DOORS;
```

Como primeira versão poderá tentar-se que o módulo faça a contagem de portas dos dois lados descritos, sendo dificil (numa primeira abordagem) para um sistema como este, baseado apenas em sensores de ultra-sons, fazê-lo de outras maneiras.

O algoritmo de contagem de portas irá basear-se na detecção de duas esquinas separadas numa determinada gama de distâncias. Assim, e analisando apenas um dos lados (para o outro é semelhante) o procedimento terá três fases:



A área 2 representa a porta que poderá estar fechada, semi-aberta ou totalmente aberta, sendo o algoritmo capaz de a detectar em qualquer uma das situações. Um problema que a segunda condição pode provocar (porta semi-aberta) é o de que se a superfície da porta fizer um ângulo superior a aproximadamente 30º em relação ao plano dos sensores, essa superfície não será 'visível' para os sensores podendo levar a conclusões falsas para futuras acções como por exemplo entrar na porta. Como o algoritmo irá analisar as esquinas, e como uma porta tem uma reentrância de pelo menos 30 cm (largura de um caixilho de uma porta), poder-se-á ter a a certeza de que se for uma porta terá pelo menos um perfil de duas esquinas separadas por uma determinada distância com uma reentrância no espaço que as separa de pelo menos 30 cm, valor este que poderá ser parametrizado.

A distância para uma porta é na maioria dos ambientes, e nomeadamente no edifício onde a plataforma se encontra, para um gabinete de 80 cm e para um laboratório de aproximadamente 100 cm (neste tipo de portas existe uma ligeira variação que ainda não parametrizámos). Com este conjunto de condições, e concentrando a análise nos três sensores laterais existentes no robot (para cada lado), poderá ser construido um identificador para o padrão porta.

5.3. Planeamento de trajectórias

Encontrando-se nesta fase do trabalho implementado, um sistema de navegação em alto nível, para o robot, tendo conhecimento do ambiente onde este se irá movimentar, poderá eventualmente partir-se daqui para o planeamento de trajectórias. Ou seja o desenvolvimento de uma aplicação, na *worstation*, que a partir do mapa do ambiente, dos pontos de partida e chegada, e de passagens intermédias (caso existam), faça o planeamento do movimento (trajectória) com base na linguagem ou nas regras já definidas. A referida aplicação poderá poderá interagir com o gestor de missões de maneira a minimizar aspectos como por exemplo a memória disponível.

Em tempo real e como o sistema permite a monotorização contínua, esta aplicação poderá determinar em qualquer instante qual o próximo passo para levar a missão a bom termo, isto é, dotar o sistema com algoritmos de aprendizagem e inteligência artificial.

6. Conclusões

Todo o trabalho desenvolvido permite concluir que utilizando apenas a informação dada pelos sensores de ultra-sons *on-board*, é possível desenvolver um sistema de navegação para o robot sem recurso a alterações do ambiente. Ou seja pode ser especificada uma missão de um ponto de partida para um ponto de chegada admitindo que o robot está numa dada posição com uma dada precisão relativa.

Durante o desenvolvimento das demonstrações efectuadas verificámos que uma condição muito importante para a escolha de zonas de calibração é que estas sejam superfícies pertencentes à estrutura intrínseca do edifício e de preferência inalteráveis. Assim, testando, por exemplo, a demonstração *demo_abril1* concluímos que existe um grande compromisso entre o nº de zonas de calibração (para ajuste da orientação do robot) e a eficácia da demonstração, pois as zonas de calibração terão necessáriamente de ser zonas de permanência tendencialmente para zero de pessoas ou objectos temporários, condição que faz com que se tenha algum cuidado em utilizá-las em grande número, para além do facto de aumentar o tempo de execução.

A estabilidade do código efectuado pode ser considerada boa (execepto no módulo *doors.c*) podendo por exemplo a demonstração *demo_abril1* ser colocada em ciclo infinito que não causará qualquer erro de execução até que as baterias do robot estejam abaixo da carga mínima. O software não detecta esta situação sendo uma futura funcionalidade que poderá ser acrescida ao módulo *emerg.c* para aumentar a fiabilidade total do sistema.

A versatilidade de todo o sistema desenvolvido até à data, incluindo todo o trabalho efectuado anteriormente, é uma das suas grandes vantagens, pois a adaptação das ferramentas desenvolvidas, conhecido o ambiente onde o robot se irá deslocar, é relativamente fácile rápida .

Finalmente, parece lícito concluir que o conjunto de funcionalidades implementadas pelo sistema desenvolvido permite cumprir os objectivos inicialmente propostos, e que naturalmente novas funcionalidades poderão ser implementadas no futuro

7. Referências

[Santos95] Vítor Manuel Ferreira dos Santos – Navegação Autónoma de Robots: Interpretação dos Dados Sensoriais e Navegação Local, Julho 1995.

[Ranger98] Emanuel Amaral de Oliveira, Paulo Miguel de Jesus Dias – Programa de Navegação e Comunicações para um Robot Móvel, relatório de Projecto, Setembro 1998.

[Robosoft97a] ROBOSOFT – Robuter™ Users´s Manual, V6.0, Março 1997.

[Robosoft97b] ROBOSOFT – Albatros™ Rerence Manual, V6.0, Março 1997.

[Robosoft97c] ROBOSOFT – Albatros™ User´s Manual, V6.0, Março 1997.

[Robosoft97d] ROBOSOFT – Albatros™ C Interface Reference Manual, V4.0, Março 97.

[Robosoft97e] ROBOSOFT – PC Albatros™ Interface, Março 1997.

8. Anexos

Anexo I

Quick User's Guide for Robuter (versão 1.0)

O robot possui um disjuntor (por baixo do lado esquerdo) que deve ser o 1º dispositivo a ser ligado. Seguidamente deve ser ligado o interruptor própriamente dito no painel, e o PC.

Da *Workstation* (máquina *Pandora*) podemos comunicar com o robot por *telnet*, visto que o PC a si acoplado se encontra ligado à rede ou através de um cabo.

- Através do cabo:

> Ligar o cabo.
> *tip -38400 /dev/ttya.*
> Ligar o robot.
> Prompt do Albatros dísponível.
> Para saír do *tip:* ~.

- Via modem:

> Ligar o PC depois do robot.
> Na sun: *telnet robuter*
> *display pandora:0*
> *xhost + robuter*
> Lançar a aplicação de terminal.
> Para saír do Kermit *CTRL\C*

- Para terminar:

Fazer 1º o shutdown do PC:
> *su root*
> *password*
> *shutdown -h now*

De seguida são descritos alguns comandos do sistema operativo que corre no robot, o ***Albatros™***, considerados os mais importantes para uma primeira abordagem evitando assim uma primeira leitura exaustiva do manual.

• Chama-se a atenção para o facto de nunca usar os comandos SERV, MOVE e MOTN quando o comando MOTV está activado ou os comandos SERV, MOVE, MOTN e MOTV, quando o comando MOTP está activado.

• O robot possui um joystick o qual pode ser usado para o mudar de sítio quando for necessário. Assim para usar o joystick:

No ***Albatros™***:

> serv off
> motv on
> motv jk on

Para deixar de usar o joystick:

> motv off
> motv jk off

• Bumpers

Sensores de contacto situados na parte da frente e na traseira do robot que se destinam essencialmente a situações extremas, tais como eminência de colisão ou evitar danos maiores em caso de colisão!.

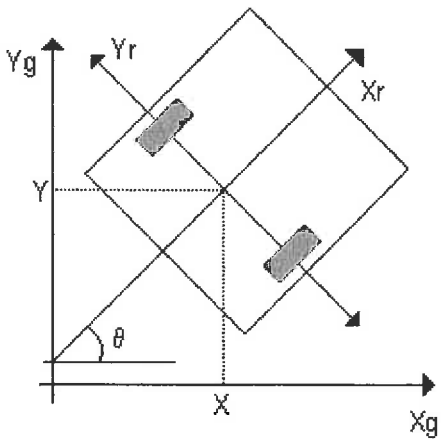
O control destes sensores é feito através do comando BUMP.

• BUMP ON | OFF (ON-activa; OFF- desactiva)

Para testes manuais é aconselhável que o sistema de bumpers esteja sempre ON.

• Sistema odométrico

O sistema odométrico fornece uma estimativa da posição num plano cartesiano, esta posição é definida por 3 coordenadas: x,y e θ, onde x e y são as coordenadas cartesianas e θ especifica a orientação do robot relativamente ao eixo dos x, como mostra a seguinte figura:



Alguns comandos:

- ODOM: Indica a posição do robot, começa e pára o processo odométrico.
- ODOS : Faz o set de um valor na odometria.
- ODOF: Faz o offset de um valor na odometria.

- **Sistema de ultra-sons**

  - READ : Leitura dos sensores.
    Exemplo: READ C = < node >, <sensors >, < range >

    onde: < node > é o nº do nodo a que o sensor pertence

    < sensors > selecção dos sensores < 1= selecionado; 0 = não selecionado.

    < range > 0 = short range; 1= longe range.

- **Alguns comandos standard**

  - **MOVE** < parametros > AC
    RC
    AR
    [D = < devices >]
    [P = < clock cycles >]
    [G = < gains arrays >]

Pârametros: P – Para movimento em posição;
V – Para movimento em velocidade;
PA – Para movimento uniformemente acelerado em posição;
PD – Para movimento uniformemente desacelerado em posição;

AC – Indica a velocidade ou a posição a alcançar;
RC – Indica posição ou velocidade relativa a alcançar;
AR – Como RC mas limpa o conteúdo dos sensores antes do movimento;
D – número do dispositivo;
P – Especifica o nº de ciclos de relógio necessários para dar ao comando para realizar o comando pretendido;

Exemplo: *move v ac=1,1* → O robot desloca-se com movimento uniforme com uma velocidade de 1 cm por segundo.

Anexo II

Mensagens interpretadas pelo módulo *serial.c*

A categoria e o tipo estão juntos e os parâmetros separados entre eles por um separador (em princípio o espaço) a mensagem é terminada por um caracter de fim de linha (\r), sendo o comprimento da mensagem variável.

A sintaxe é a seguinte:

```
<mensagem recebida> ::= <categoria> <tipo> [<parâmetros>] <terminador>
<mensagem enviada>  ::= <tipo> <informação> <terminador>
<parâmetros>        ::= <parâmetro> <separador> <parâmetro> …
<informação>        ::= <informação> <separador> <informação> …
<terminador>        ::= <fim de linha (/r)>
<separador>         ::= <espaço> ou <virgula>
```

Mensagens de pedido de informação

São mensagens que começam com o caracter B e que pedem informações ao robot, pelo que esperam uma resposta.

#Define	Char	Descrição e exemplo	Exemplo de resposta
IN_POST	'O'	Pedido da postura do robot **BO**	S 200 120 34 1 162341
IN_US	'U'	Pedido dos dados dos ultra-sons **BU**	U 750 1232 … 162341
IN_POSTUS	'V'	Pedido da postura e dados dos ultra--sons **BV**	V 200 120 34 1 750 … 162341
IN_CLOCK	'C'	Pedido do relógio interno do robot **BC**	C 162341
IN_USTOUT	'T'	Pede o valor do *Time out* na leitura dos ultra-sons **BT**	T15
IN_PTU_US	'P'	Pedido os dados da *Pan & Tilt* **BP**	P 2491 2491 ... 750 162341
IN_PTU_FUS	'F'	Pedido dos dados filtrados da *Pan & Tilt* **BF**	U 2491 2491 ... 750 162341
IN_PTU_DUS	'D'	Pedido das derivadas dos valores medidos na *Pan & Tilt* **BV**	V 0 0 200 –300 ...750 162341

Mensagens unidireccionais

São mensagens que começam com a letra A e que enviam um comando para o programa efectuar. São respondidas com um *ack* ou um *error*.

#Define	Char	Descrição e exemplo	Exemplo de comandos
IN_VELOCITY	'M'	Impõe um movimento em velocidade (cm/s) **AM <esquerda> <direita>**	AM 5 4
IN_SERV	'S'	Activa/desactiva o processo de baixo nível de controlo de velocidade **AS<estado> N:inactivo F: activo**	ASN
IN_ALBATROS	'Y'	Comando directo ao Albatros **AY<comando>**	AYODOM ON
IN_ODOM	'C'	Impõe a postura do robot **AC<postura>**	AC0 0 0
IN_POSITION	'W'	Impõe um movimento em posição. **AW <x y theta>**	AW 300 250 10
IN_KILL	'X'	Termina a aplicação	AX
IN_PTU_ON	'p'	Ligar *Pan & Tilt* **Ap**	Ap
IN_PTU_OFF	'g'	Desligar *Pan & Tilt* **Ag**	Ag
IN_GATE	'G'	Ligar módulo de busca de porta **AG**	AG
IN_END_GATE	'T'	Desligar o módulo de busca de porta **AT**	AT
IN_DOOR	'd	Ligar modo de passagem de portas **Ad**	Ad
IN_DOOR_OFF	'f'	Desligar modo de passagem de portas **Af**	Af

Mensagens associados aos ultra-sons

São mensagens que começam com a letra U e que permitem pedir informações (exigindo neste caso uma resposta) ou enviar configurações (respondidas só pelo *acknowledge*).

#Define	Char	Descrição e exemplo	Exemplo de comando
IN_USTABLE	'T'	Configura a tabela de sensores activos **UT<tabela>**	UT101...1001
IN_NDELAY	'D'	Configura o tempo de disparo entre nodos **UD<tempo>**	UD5
IN_GNDELAY	'N'	Pede o tempo de disparo entre nodos **UN**	UN
IN_STAT	'S'	Pede informações estatísticas relativas aos ultra-sons: **USN**: tempo médio para ler um sensor (em ticks) **USS**: tempo médio para ler os sensores todos (em ticks)	S45
IN_SUSTOUT	't'	Impõe o time_out **Ut<time_out>**	Ut30
IN_TOGSENS	'u'	Modo automático-desliga sensores inúteis **Uu< 0: inactivo 1: activo>**	Uu1

Respostas

A seguir são apenas indicadas as respostas específicas, uma mensagem que só configura o sistema e não espera resposta específica deve ser sempre respondida com um *ack* ou *error*.

Todas as resposta terminam com o terminador fim de linha.

#Define	Char	Descrição e exemplo	Resposta a
OUT_POST	'S'	Postura do robot **S <x> <y> <θ> <s> <t>**	BO
OUT_US	'U'	Dados dos ultra-sons **U <u1>...<u24> <s> <t>**	BU
OUT_POSTUS	'V'	Postura e dados dos ultra-sons **S <x> <y> <θ> <u1>...<u24> <s> <t>**	BV
OUT_CLOCK	'C'	Relógio interno do robot **C <relógio>**	BC
OUT_GDEL	'N'	Tempo de disparo entre nodos **N <tempo>**	UN

Anexo III

Manual de edição e gestão de acções

Para o envio de comandos existe um formato para as mensagens:

|category| |Type| |Parameter 1| |Parameter 2|.....

A tabela seguinte indica os comandos possiveis na categoria C:

C Category (*isolated commands for mission management*)		
Type	*Description*	*Example*
A	Start the mission processing	CA
S	Stop the mission processing	CS
P	Pause	CP
E	Clear the task mission array	CE
D	Delete a task in mission array	CD 25
R	Replace an exintent task	CR 12
I	Insert a task in mission array	CI 10 3 100 20 90
L	Give the list of tasks to execute	CL
T	Manual tasking processing	CT 4 100 90 20

Na categoria C existe a possibilidade da execução de tarefas isoladas através do comando CT, o protocolo utilizado está descrito na tabela da página seguinte:

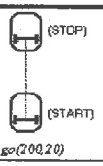
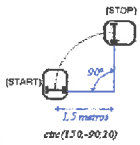
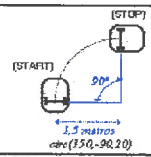
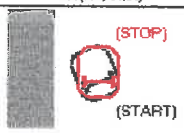
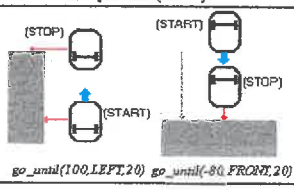
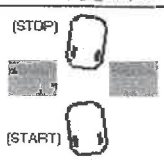
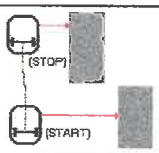
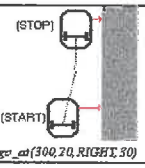
Exemplo:

Tarefa:

go(100,30); / Deslocar-se em frente durante 1 metro, a 30 cm/s */*

Mensagem:

CT 1 100 30

Protocol to edit a task (manual or in download mode)						
Task name	*type*	*Parameter 1*	*Parameter 2*	*Parameter 3*	*Parameter 4*	*Example*
go	1	*distance (cm)*	*velocity (cm/s)*	-	-	
roll	2	*ângulo (º)*	*velocity*	-	-	
curve	3	*distance (cm)*	*ângulo (º)*	*velocity(cm/s)*	-	
circ	4	*ray (cm)*	*ângulo (º)*	*velocity(cm/s)*	-	
parallel	5	*Side*	-	-	-	
go_until	6	*distance (cm)*	*Side*	*velocity(cm/s)*	-	
cross_door	7	-	-	-	-	
go_parallel	8	*distance (cm)*	*Side*	*velocity(cm/s)*	-	
go_at	9	*distance (cm)*	*lateral (cm)*	*Side*	*velocity(cm/s)*	
active_us	10	*sensor 0 to 7*	*sensor 8 to 15*	*sensor 16 to 23*	*emerg.(ON/OFF)*	*active_us(11000111,00011100,01110000,1)*

Anexo IV

kernel.c

mission_executor.h
mission_executor.c

serial.h
serial.c

doors.h
doors.c

ptu.h
ptu.c

emerg.h
emerg.c

us.h
us.c

executor.h
executor.c

detect.h
detect.c

headers.h

generic.h
generic.c

```
/* S.N.A.N. para RobuterIII

  Anabela Duarte  &  Paulo Peixoto

*****************************************************************/

#include <libcstd.h>
#include <libcalbl.h>
#include <libcpro.h>

#include "us.h"
#include "serial.h"
#include "generic.h"
#include "executor.h"
#include "emerg.h"
#include "ptu.h"
#include "doors.h"
#include "detect.h"
#include "headers.h"
#include "mission_executor.h"


int kill=FALSE;

/*                  period,  priority,  offset */
Procs_To_Add tab1={8, read_sensors,        4,          2,        6,
                     mission_executor,   40,          2,        50,
                     detect,             70,          2,        50,
                     serial_spy,          2,          6,        10,
                     execute,            10,          4,        2,
                     doors,               2,          4,        50,
                     ptu,                 2,          5,        50,
                     emergency,          40,          7,        10 };

Procs_To_Remove tab2={8, read_sensors, mission_executor, detect, serial_spy, execute, doors, ptu, emergency };


void main()
{
/* Processes initializations */

  init_emergency();
  init_execute();
  init_serial_spy();
  init_read_sensors();
  init_ptu();
  init_doors();
  init_navigator();

  kill=FALSE;

  init_procit();

  while(add_procit(&tab1));

  while(kill==FALSE);

  printf("\nTerminar processos\n");
  while(rem_procit(&tab2));
  close_execute();
  close_ptu();
  close_serial_spy();
}
```

```
/* S.N.A.N. para RobuterIII

  Anabela Duarte  &  Paulo Peixoto

*****************************************************************/

#define DISTX 2500                                  /* old values  of 98 demo      */
#define DISTY 10000                                 /* old values  of 98 demo      */
#define RAIO 10000                                  /* old values  of 98 demo      */

#define T_GO            1                           /* type of mission tasks (parameter[0]) */
#define T_ROLL          2                           /* type of mission tasks (parameter[0]) */
#define T_CURVE         3                           /* type of mission tasks (parameter[0]) */
#define T_CIRC          4                           /* type of mission tasks (parameter[0]) */
#define T_PARALLEL      5                           /* type of mission tasks (parameter[0]) */
#define T_GO_UNTIL      6                           /* type of mission tasks (parameter[0]) */
#define T_CROSS_DOOR    7                           /* type of mission tasks (parameter[0]) */
#define T_GO_PARALLEL   8                           /* type of mission tasks (parameter[0]) */
#define T_GO_AT         9                           /* type of mission tasks (parameter[0]) */


#define FRONT  1
#define RIGHT  2
#define BACK   3
#define LEFT   4


void mission_executor(void);                                    /* prototype of mission_executor */
void init_navigator(void);                                      /* prototype of init_navigator  */
int roll(long int theta_in, int velocity_in);                   /* prototype of roll            */
int go(int dist,int v_in);                                      /* prototype of go              */
int circ(float raio, long int theta_in, int v_in);              /* prototype of circ            */
int curve(float l,long int theta_in, int v_in);                 /* prototype of curve                    */
int parallel(int side);                                         /* prototype of parallel        */
int go_until(int dist,int side,int v_in);                       /* prototype of go_until        */
int cross_door(void);                                           /* prototype of cross_door      */
int go_parallel(int dist,int side,int v_in);                    /* prototype of go_parallel     */
int go_at(int dist_f,int dist_wall,int side,int v_in);          /* prototype of go_at                    */
void active_us(int seccao1, int seccao2, int seccao3, int emg); /* prototype of active_us */
void reset_tasks_states(void);                                  /* prototype of reset_tasks_states */

/*****************************************************************************/


typedef struct {
        int state;
        int mission_state;
        int type;
        int parameter1;
        int parameter2;
        int parameter3;
        int parameter4;
        } S_MANUAL_TYPE;

/*****************************************************************************/

typedef struct {
        int state;                                      /* estado da funcao roll        */
        long int old_x;                                 /* valor inicial de x           */
        long int old_y;                                 /* valor inicial de y           */
        long int old_theta;                             /* valor inicial de theta       */
        int velocity;                                   /* velocidade (media)           */
        int erro;                                       /* erro associado ao movimento  */
        } S_ROLL;

typedef struct {
        int state;                                      /* estado da funcao go          */
        long int old_x;                                 /* valor inicial de x           */
        long int old_y;                                 /* valor inicial de y           */
        long int old_theta;                             /* valor inicial de theta       */
        int v;                                          /* velocidade (media)           */
        int erro;                                       /* erro associado ao movimento  */
        } S_GO;

typedef struct {
        int state;                                      /* estado da funcao circ        */
        long int old_x;                                 /* valor inicial de x           */
        long int old_y;                                 /* valor inicial de y           */
        long int old_theta;                             /* valor inicial de theta       */
        float vr;                                       /* velocidade (motor direito)   */
        float vl;                                       /* velocidade (motor esquerdo)  */
        int erro;                                       /* erro associado ao movimento  */
        } S_CIRC;

typedef struct {
        int state;                                      /* estado da funcao curve       */
        long int old_x;                                 /* valor inicial de x           */
        long int old_y;                                 /* valor inicial de y           */
        long int old_theta;                             /* valor inicial de theta       */
        float vr;                                       /* velocidade (motor direito)   */
        float vl;                                       /* velocidade (motor esquerdo)  */
        int erro;                                       /* erro associado ao movimento  */
        } S_CURVE;

typedef struct {
        int state;                                      /* estado da funcao curve       */
        int erro;                                       /* erro associado ao movimento  */
        } S_PARALLEL;

typedef struct {
        int state;                                      /* estado da funcao go_until    */
        long int old_x;                                 /* valor inicial de x           */
        long int old_y;                                 /* valor inicial de y           */
        long int old_theta;                             /* valor inicial de theta       */
        int v;                                          /* velocidade (media)           */
        int erro;                                       /* erro associado ao movimento  */
        int xsensor;                                    /* sensor perpendicular a direccao */
        } S_GOU;
```

```
typedef struct {
        int state;                              /* estado da funcao cross_door */
        } S_CROSS_DOOR;

typedef struct {
        int state;                              /* estado da funcao go_parallel          */
        long int old_x;                         /* valor inicial de x            */
        long int old_y;                         /* valor inicial de y            */
        long int old_theta;                     /* valor inicial de theta        */
        int vr;                                 /* velocidade (roda direita)     */
        int vl;                                 /* velocidade (roda esquerda)    */
        int erro;                               /* erro associado ao movimento   */
        } S_GO_PARALLEL;

typedef struct {
        int state;                              /* estado da funcao go_at        */
        long int old_x;                         /* valor inicial de x            */
        long int old_y;                         /* valor inicial de y            */
        long int old_theta;                     /* valor inicial de theta        */
        int vr;                                 /* velocidade (roda direita)     */
        int vl;                                 /* velocidade (roda esquerda)    */
        int erro;                               /* erro associado ao movimento   */
        } S_GO_AT;

/*****************************************************************************/
```

```
/* S.N.A.N. para RobuterIII

  Anabela Duarte  &  Paulo Peixoto

*****************************************************************/

#include "libcstd.h"
#include "libcalbl.h"
#include "generic.h"
#include "us.h"
#include "executor.h"
#include "emerg.h"
#include "serial.h"
#include "headers.h"
#include "detect.h"

#include "mission_executor.h"

extern int kill;
extern US_DATA sens;
extern POSTURE pos;
extern MOVES mov;
extern EMERGENCY emerg;

extern HISTORY hist;
extern int door;
extern int auto_detect;
extern int success;


extern S_MISSION_TASK mission_array[MAX_TASKS];
extern int mission_manager[MAX_TASKS];
extern int mission_state;
extern int position_array;
extern int start_mission;

S_MANUAL_TYPE f_manual_type;

S_ROLL f_roll;
S_GO f_go;
S_CIRC f_circ;
S_CURVE f_curve;
S_PARALLEL f_parallel;
S_GOU f_gou;
S_CROSS_DOOR f_cross_door;
S_GO_PARALLEL f_go_parallel;
S_GO_AT f_go_at;
int mission_state;
/*****************************************************************/

void mission_executor(void)
{
        int keep_mission_state;

        if(start_mission==0)
                if(f_manual_type.state>0)
                   switch(f_manual_type.state)
                        {
                           case 1:
                                   f_manual_type.mission_state=mission_state;
                                   f_manual_type.state=2;
                                   break;
                           case 2:
                                   switch(f_manual_type.type)
                                       {
                                               case 1:
                                                       go(f_manual_type.parameter1,f_manual_type.parameter2);
                                                       break;
                                               case 2:
                                                       roll(f_manual_type.parameter1,f_manual_type.parameter2);
                                                       break;
                                               case 3:
curve(f_manual_type.parameter1,f_manual_type.parameter2,f_manual_type.parameter3);
                                                       break;
                                               case 4:
circ(f_manual_type.parameter1,f_manual_type.parameter2,f_manual_type.parameter3);
                                                       break;
                                               case 5:
                                                       parallel(f_manual_type.parameter1);
                                                       break;
                                               case 6:
go_until(f_manual_type.parameter1,f_manual_type.parameter2,f_manual_type.parameter3);
                                                       break;
                                               case 7:
                                                       cross_door();
                                                       break;
                                               case 8:
go_parallel(f_manual_type.parameter1,f_manual_type.parameter2,f_manual_type.parameter3);
                                                       break;
                                               case 9:
go_at(f_manual_type.parameter1,f_manual_type.parameter2,f_manual_type.parameter3,f_manual_type.parameter4);
                                                       break;
                                               case 10:
active_us(f_manual_type.parameter1,f_manual_type.parameter2,f_manual_type.parameter3,f_manual_type.parameter4);
                                                       break;
                                        };
                                   if(f_manual_type.mission_state != mission_state)
                                       {
                                               f_manual_type.parameter1=0;
                                               f_manual_type.parameter2=0;
                                               f_manual_type.parameter3=0;
                                               f_manual_type.parameter4=0;
```

```
                                        mission_state=f_manual_type.mission_state;
                                        f_manual_type.mission_state=0;
                                        f_manual_type.state=0;
                                    };
                                break;
                        };

        keep_mission_state=mission_state;

           if(start_mission==1)
                if(mission_array[position_array].state==1)
                  switch(mission_array[position_array].parameter[0])
                        {
                                case T_GO:
go(mission_array[position_array].parameter[1],mission_array[position_array].parameter[2]);
                                        if(keep_mission_state != mission_state)
                                                {
                                                        mission_array[position_array].parameter[0]=0;
                                                        mission_array[position_array].parameter[1]=0;
                                                        mission_array[position_array].parameter[2]=0;
                                                        mission_array[position_array].state=2;
                                                        position_array=mission_array[position_array].next;
                                                };
                                        break;
                                case T_ROLL:
roll(mission_array[position_array].parameter[1],mission_array[position_array].parameter[2]);
                                        if(keep_mission_state != mission_state)
                                                {
                                                        mission_array[position_array].parameter[0]=0;
                                                        mission_array[position_array].parameter[1]=0;
                                                        mission_array[position_array].parameter[2]=0;
                                                        mission_array[position_array].state=2;
                                                        position_array=mission_array[position_array].next;
                                                };
                                        break;
                                case T_CURVE:
curve(mission_array[position_array].parameter[1],mission_array[position_array].parameter[2],mission_array[position_array].paramet
                                        if(keep_mission_state != mission_state)
                                                {
                                                        mission_array[position_array].parameter[0]=0;
                                                        mission_array[position_array].parameter[1]=0;
                                                        mission_array[position_array].parameter[2]=0;
                                                        mission_array[position_array].parameter[3]=0;
                                                        mission_array[position_array].state=2;
                                                        position_array=mission_array[position_array].next;
                                                };
                                        break;
                                case T_CIRC:
circ(mission_array[position_array].parameter[1],mission_array[position_array].parameter[2],mission_array[position_array].paramete
                                        if(keep_mission_state != mission_state)
                                                {
                                                        mission_array[position_array].parameter[0]=0;
                                                        mission_array[position_array].parameter[1]=0;
                                                        mission_array[position_array].parameter[2]=0;
                                                        mission_array[position_array].parameter[3]=0;
                                                        mission_array[position_array].state=2;
                                                        position_array=mission_array[position_array].next;
                                                };
                                        break;

                                case T_PARALLEL:
                                        parallel(mission_array[position_array].parameter[1]);
                                        if(keep_mission_state != mission_state)
                                                {
                                                        mission_array[position_array].parameter[0]=0;
                                                        mission_array[position_array].parameter[1]=0;
                                                        mission_array[position_array].state=2;
                                                        position_array=mission_array[position_array].next;
                                                };
                                        break;
                                case T_GO_UNTIL:
go_until(mission_array[position_array].parameter[1],mission_array[position_array].parameter[2],mission_array[position_array].para
                                        if(keep_mission_state != mission_state)
                                                {
                                                        mission_array[position_array].parameter[0]=0;
                                                        mission_array[position_array].parameter[1]=0;
                                                        mission_array[position_array].parameter[2]=0;
                                                        mission_array[position_array].parameter[3]=0;
                                                        mission_array[position_array].state=2;
                                                        position_array=mission_array[position_array].next;
                                                };
                                        break;
                                case T_CROSS_DOOR:
                                        cross_door();
                                        if(keep_mission_state != mission_state)
                                                {
                                                        mission_array[position_array].parameter[0]=0;
                                                        mission_array[position_array].state=2;
                                                        position_array=mission_array[position_array].next;
                                                };
                                        break;
                                case T_GO_PARALLEL:
go_parallel(mission_array[position_array].parameter[1],mission_array[position_array].parameter[2],mission_array[position_array].p
                                        if(keep_mission_state != mission_state)
                                                {
                                                        mission_array[position_array].parameter[0]=0;
                                                        mission_array[position_array].parameter[1]=0;
                                                        mission_array[position_array].parameter[2]=0;
                                                        mission_array[position_array].parameter[3]=0;
                                                        mission_array[position_array].state=2;
                                                        position_array=mission_array[position_array].next;
                                                };
```

```
                                                        break;                                                                  205
                                                case T_GO_AT:
                                                                                                                                ...
go_at(mission_array[position_array].parameter[1],mission_array[position_array].parameter[2],mission_array[position_array].paramet
                                                        if(keep_mission_state != mission_state)
                                                                {
                                                                        mission_array[position_array].parameter[0]=0;
                                                                        mission_array[position_array].parameter[1]=0;
                                                                        mission_array[position_array].parameter[2]=0;
                                                                        mission_array[position_array].parameter[3]=0;
                                                                        mission_array[position_array].parameter[4]=0;
                                                                        mission_array[position_array].state=2;
                                                                        position_array=mission_array[position_array].next;
                                                                };
                                                        break;
                                                case T_ACTIVE_US:
                                                                                                                                ...
active_us(mission_array[position_array].parameter[1],mission_array[position_array].parameter[2],mission_array[position_array].par
                                                        if(keep_mission_state != mission_state)
                                                                {
                                                                        mission_array[position_array].parameter[0]=0;
                                                                        mission_array[position_array].parameter[1]=0;
                                                                        mission_array[position_array].parameter[2]=0;
                                                                        mission_array[position_array].parameter[3]=0;
                                                                        mission_array[position_array].parameter[4]=0;
                                                                        mission_array[position_array].state=2;
                                                                        position_array=mission_array[position_array].next;
                                                                };
                                                        break;
                                        };
                        if(mission_state>101)
                           {
                                mov.left=0;
                                mov.right=0;
                                mission_state=0;
                                kill=TRUE;
                           };
                        emerg.rtm=TRUE;
}; /* end of mission_executor */
/*****************************************************************************/
void reset_tasks_states(void)
{
        f_go.state=0;
        f_roll.state=0;
        f_circ.state=0;
        f_curve.state=0;
        f_parallel.state=0;
        f_gou.state=0;
        f_cross_door.state=0;
        f_go_parallel.state=0;
        f_go_at.state=0;

}; /*end of reset_tasks_states */
/*****************************************************************************/
void active_us(int seccao1, int seccao2, int seccao3, int emg)
{
        int s[24];
        int seccao;
        int i;

        seccao=seccao1;
        s[0]=seccao/10000000;
        seccao=seccao % 10000000;
        s[1]=seccao/1000000;
        seccao=seccao % 1000000;
        s[2]=seccao/100000;
        seccao=seccao % 100000;
        s[3]=seccao/10000;
        seccao=seccao % 10000;
        s[4]=seccao/1000;
        seccao=seccao % 1000;
        s[5]=seccao/100;
        seccao=seccao % 100;
        s[6]=seccao/10;
        seccao=seccao % 10;
        s[7]=seccao;
        seccao=seccao2;
        s[8]=seccao/10000000;
        seccao=seccao % 10000000;
        s[9]=seccao/1000000;
        seccao=seccao % 1000000;
        s[10]=seccao/100000;
        seccao=seccao % 100000;
        s[11]=seccao/10000;
        seccao=seccao % 10000;
        s[12]=seccao/1000;
        seccao=seccao % 1000;
        s[13]=seccao/100;
        seccao=seccao % 100;
        s[14]=seccao/10;
        seccao=seccao % 10;
        s[15]=seccao;
        seccao=seccao3;
        s[16]=seccao/10000000;
        seccao=seccao % 10000000;
        s[17]=seccao/1000000;
        seccao=seccao % 1000000;
        s[18]=seccao/100000;
        seccao=seccao % 100000;
        s[19]=seccao/10000;
        seccao=seccao % 10000;
        s[20]=seccao/1000;
        seccao=seccao % 1000;
        s[21]=seccao/100;
        seccao=seccao % 100;
        s[22]=seccao/10;
        seccao=seccao % 10;
        s[23]=seccao;
```

```
        if(emg < 1)
                {
                        emerg.on=BUMPER;
                }
                else
                        {
                                emerg.on=BUMPER|US;
                        };

        for(i=0;i<=23;i++)
                {
                        if(s[i] >=1)
                                {
                                        s[i]=1;
                                }
                                else
                                   {
                                        s[i]=0;
                                   };
                        sens.active_sensors[i]=s[i];
                };
        sens.change_flag=TRUE;

        mission_state++;

}; /* end of active_us */
/*****************************************************************/
void init_navigator(void)
{
        mission_state=0;
        position_array=0;
        start_mission=0;

        f_cross_door.state=0;    /* inicialization of cross_door function */
        auto_detect=OFF;

        f_manual_type.state=0;          /* inictialization of manual type*/
        f_manual_type.mission_state=0;
        f_manual_type.type=0;
        f_manual_type.parameter1=0;
        f_manual_type.parameter2=0;
        f_manual_type.parameter3=0;
        f_manual_type.parameter4=0;

        f_roll.state=0;          /* inictialization of roll function */
        f_roll.old_x=0;
        f_roll.old_y=0;
        f_roll.old_theta=0;
        f_roll.velocity=0;
        f_roll.erro=0;

        f_go.state=0;            /* inictialization of go function */
        f_go.old_x=0;
        f_go.old_y=0;
        f_go.old_theta=0;
        f_go.v=0;
        f_go.erro=0;

        f_circ.state=0;          /* inictialization of circ function */
        f_circ.old_x=0;
        f_circ.old_y=0;
        f_circ.old_theta=0;
        f_circ.vr=0;
        f_circ.vl=0;
        f_circ.erro=0;

        f_curve.state=0;         /* inictialization of curve function */
        f_curve.old_x=0;
        f_curve.old_y=0;
        f_curve.old_theta=0;
        f_curve.vr=0;
        f_curve.vl=0;
        f_curve.erro=0;

        f_parallel.state=0;      /* inictialization of parallel function */
        f_parallel.erro=2;

        f_gou.state=0;           /* inictialization of go_until function */
        f_gou.old_x=0;
        f_gou.old_y=0;
        f_gou.old_theta=0;
        f_gou.v=0;
        f_gou.erro=0;
        f_gou.xsensor=1;

        f_cross_door.state=0;    /* inictialization of cross_door function */

        f_go_parallel.state=0;          /* inicialization of go_parallel function */
        f_go_parallel.old_x=0;
        f_go_parallel.old_y=0;
        f_go_parallel.old_theta=0;
        f_go_parallel.vr=0;
        f_go_parallel.vl=0;
        f_go_parallel.erro=0;

        f_go_at.state=0;                /* inictialization of go_at function */
        f_go_at.old_x=0;
        f_go_at.old_y=0;
        f_go_at.old_theta=0;
        f_go_at.vr=0;
        f_go_at.vl=0;
        f_go_at.erro=0;


}; /* end of init_navigator() */
/*****************************************************************************/
```

```
int go_at(int dist_f,int dist_wall,int side,int v_in)
{
        int sens_1,sens_2;
        int step=1;

        dist_f=dist_f*100;
        dist_wall=dist_wall*10;

        if(v_in>=100)           /* limit of v_in */
           v_in=100;
        if(v_in<=-100)          /* limit of v_in */
                v_in=0;

        if(dist_f<0)
                {
                     dist_f=-dist_f;
                     printf("\nWarning (bad argument in go_at (dist_f<0) ");
                 };
        if(f_go_at.state==0)
                {
                        f_go_at.old_x=pos.x;
                        f_go_at.old_y=pos.y;
                        f_go_at.old_theta=pos.theta;
                        sendf("ODOS C=%ld,%ld,%ld",0,0,0);  /* reset dos encoders */
                        pos.x=0;
                        pos.y=0;
                        pos.theta=0;
                        f_go_at.erro=v_in/10;
                        f_go_at.vl=v_in;
                        f_go_at.vr=v_in;
                        f_go_at.state=1;
                                        };                                  /* end of state 0 */
        if(f_go_at.state==1)
                {
                        if (side==RIGHT)
                                {
                                        sens_1=5;
                                        sens_2=7;
                                        if(sens.values[sens_1]>dist_wall)
                                                {
                                                        mov.left=v_in+step;
                                                        mov.right=v_in-step;
                                                }
                                                else
                                                        {
                                                                mov.left=v_in-step;
                                                                mov.right=v_in+step;
                                                        };
                                }; /* end of RIGHT */
                        if (side==LEFT)
                                {
                                        sens_1=17;
                                        sens_2=19;
                                        if(sens.values[sens_2]<dist_wall)
                                                {
                                                        mov.left=v_in+step;
                                                        mov.right=v_in-step;
                                                }
                                                else
                                                   {
                                                        mov.left=v_in-step;
                                                        mov.right=v_in+step;
                                                   };
                                }; /* end of LEFT */

                        if(v_in>0)
                           if(pos.x>= dist_f)
                                {
                                    mov.left=0;
                                    mov.right=0;
                                     mission_state++;
                                    f_go_at.vr=0;
                                    f_go_at.vl=0;
                                    sendf("ODOS C=%ld,%ld,%ld"
,pos.x+f_go_at.old_x,pos.y+f_go_at.old_y,pos.theta+f_go_at.old_theta);
                                    f_go_at.state=0;
                                };
                }; /* end of state 1 */
        return(f_go_at.state);
}; /* end of go_at function */
/*****************************************************************************/
int go_parallel(int dist,int side,int v_in)
{
        int aux=0;
        int sens_1,sens_2;
        int step=1;

        dist=dist*100;

        if(v_in>=100)           /* limit of v_in */
           v_in=100;
        if(v_in<=-100)          /* limit of v_in */
                v_in=0;

        if(dist<0)
                {
                     dist=-dist;
                     printf("\nWarning (bad argument in go (dist<0) ");
                 };
        if(f_go_parallel.state==0)
                {
                        f_go_parallel.old_x=pos.x;
                        f_go_parallel.old_y=pos.y;
                        f_go_parallel.old_theta=pos.theta;
                        sendf("ODOS C=%ld,%ld,%ld",0,0,0);  /* reset dos encoders */
                        pos.x=0;
                        pos.y=0;
                        pos.theta=0;
```

```
                        f_go_parallel.erro=v_in/10;                                                    526
                        f_go_parallel.vl=v_in;
                        f_go_parallel.vr=v_in;
                        f_go_parallel.state=1;
                                        };                                              /* end of state 0 */
        if(f_go_parallel.state==1)
                {
                        if(dist>20000)
                                aux=3000;
                                        else
                                                aux=2000;
                        if (side==RIGHT)
                                {
                                        sens_1=5;
                                        sens_2=7;
                                        if(sens.values[sens_1]>sens.values[sens_2])
                                                {
                                                                mov.left=v_in+step;
                                                                mov.right=v_in-step;
                                                }
                                                else
                                                        {
                                                                mov.left=v_in-step;
                                                                mov.right=v_in+step;
                                                        };
                                }; /* end of RIGHT */
                        if (side==LEFT)
                                {
                                        sens_1=17;
                                        sens_2=19;
                                        if(sens.values[sens_1]>=sens.values[sens_2])
                                                {
                                                                mov.left=v_in+step;
                                                                mov.right=v_in-step;
                                                }
                                                 else
                                                     {
                                                                mov.left=v_in-step;
                                                                mov.right=v_in+step;
                                                      };
                                }; /* end of LEFT */

                        if(v_in>0)
                           if(pos.x>= dist)
                                {
                                    mov.left=0;
                                    mov.right=0;
                                    f_go_parallel.vr=0;
                                    f_go_parallel.vl=0;
                                    sendf("ODOS C=%ld,%ld,%ld"                                                                  ...
,pos.x+f_go_parallel.old_x,pos.y+f_go_parallel.old_y,pos.theta+f_go_parallel.old_theta);
                                    f_go_parallel.state=0;
                                    mission_state++;
                                };
                }; /* end of state 1 */
        return(f_go_parallel.state);
}; /* end of go_parallel function */

/*****************************************************************************/

int cross_door(void)
{
        if(f_cross_door.state==0)
                {
                        auto_detect=ON;
                        f_cross_door.state=1;
                        mov.left=20;
                        mov.right=20;
                };
        if(success==TRUE)
                {
                        mov.left=0;
                        mov.right=0;

                        f_cross_door.state=0;
                        mission_state++;
                        auto_detect=OFF;
                        success=FALSE;
                };

        return(f_cross_door.state);
}; /* end of cross_door */
/*****************************************************************************/
/* movimento que percorre a direccao actual ate a uma distancia x do lado pretendido
  com uma velocidade v                                                          */

int go_until(int dist, int side, int v_in)
{


        dist=dist*10;


        if(v_in>80)             /* limit of v_in */
           v_in=80;
        if(v_in<-80)
           v_in=-80;
        if(f_gou.state==0)
                {
                        f_go.old_x=pos.x;
                        f_go.old_y=pos.y;
                        f_go.old_theta=pos.theta;
                        sendf("ODOS C=%ld,%ld,%ld",0,0,0);   /* reset dos encoders */
                        pos.x=0;
                        pos.y=0;
                        pos.theta=0;
                        if (side==FRONT)
                                f_gou.xsensor=1;
```

```
                        if (side==RIGHT)
                                f_gou.xsensor=6;
                        if (side==BACK)
                                f_gou.xsensor=12;
                        if (side==LEFT)
                                f_gou.xsensor=18;
                        f_gou.erro=v_in/10;
                        f_gou.state=1;
                };  /* end of state 0 */
        if(f_gou.state==1)
                {
                        if(v_in>0)
                                {
                                        f_gou.v=v_in;
                                        if( pos.x < 200)
                                                f_gou.v/=3;
                                        if( pos.x < 50)
                                                f_gou.v/=5;
                                        if(f_gou.v<3)
                                            f_gou.v=3;
                                };
                        if(v_in<0)
                                {
                                        f_gou.v= -v_in;
                                        if(-pos.x<200)
                                            f_gou.v/=3;
                                        if(-pos.x<50)
                                            f_gou.v/=5;
                                        if(f_gou.v<3)
                                            f_gou.v=3;
                                        f_gou.v= -f_gou.v;
                                };
                        mov.left=f_gou.v;
                        mov.right=f_gou.v;

                    if(dist<0)
                        {
                        if((side==FRONT) || (side==BACK))
                                {
                                        dist=dist+100;                  /* correcao das distancias */
                                        if(sens.values[f_gou.xsensor]<= -(dist-300))
                                                {
                                                        mov.left=f_gou.v/2;
                                                        mov.right=f_gou.v/2;
                                                };
                                        if(sens.values[f_gou.xsensor]<= -(dist-100))
                                                {
                                                        mov.left=f_gou.v/3;
                                                        mov.right=f_gou.v/3;
                                                };
                                };
                        if(sens.values[f_gou.xsensor]<= -dist)
                                {
                                   mov.left=0;                  /* stop */
                                   mov.right=0;
                                   f_gou.v=0;
                                   sendf("ODOS C=%ld,%ld,%ld",pos.x+f_go.old_x,pos.y+f_go.old_y,pos.theta+f_go.old_theta);
                                   f_gou.state=0;
                                   f_gou.erro=0;
                                   mission_state++;
                                };
                        }; /* end of dist<0 */
                    if(dist>0)
                        {
                        if((side==FRONT) || (side==BACK))
                                {
                                        dist=dist-250;          /* correcao das distancias */
                                        if(sens.values[f_gou.xsensor]>= (dist-300))
                                                {
                                                        mov.left=f_gou.v/2;
                                                        mov.right=f_gou.v/2;
                                                };
                                        if(sens.values[f_gou.xsensor]>= (dist-100))
                                                {
                                                        mov.left=f_gou.v/3;
                                                        mov.right=f_gou.v/3;
                                                };
                                };
                        if(sens.values[f_gou.xsensor]>= dist)
                                {
                                   mov.left=0;                  /* stop */
                                   mov.right=0;
                                   f_gou.v=0;
                                   sendf("ODOS C=%ld,%ld,%ld",pos.x+f_go.old_x,pos.y+f_go.old_y,pos.theta+f_go.old_theta);
                                   f_gou.state=0;
                                   f_gou.erro=0;
                                   mission_state++;
                                };

                        }; /* end of dist>0 */
                };  /* end of state 1 */
        return(f_gou.state);
}; /* end of go_until function */
/*****************************************************************************/
/* orientacao do robot paralelamente ao lado  indicado */

int parallel(int side)
{
        int sens_1,sens_2;
        long dif;

        if(f_parallel.state==0)
                {
                        printf("\n Inicio da calibracao \n");
                        f_parallel.state=1;
                }; /* end of state 0 */

        if(f_parallel.state==1)
```

```
                {
                        if (side==FRONT)
                                {
                                        sens_1=0;
                                        sens_2=23;
                                        if(sens.values[sens_1]>sens.values[sens_2])
                                                {
                                                        mov.left=-1;
                                                        mov.right=1;
                                                }
                                                else
                                                        {
                                                                mov.left=1;
                                                                mov.right=-1;
                                                        };
                                }; /* end of FRONT */
                        if (side==RIGHT)
                                {
                                        sens_1=5;
                                        sens_2=7;
                                        if(sens.values[sens_1]>sens.values[sens_2])
                                                {
                                                        mov.left=1;
                                                        mov.right=-1;
                                                }
                                                else
                                                        {
                                                                mov.left=-1;
                                                                mov.right=1;
                                                        };
                                }; /* end of RIGHT */
                        if (side==BACK)
                                {
                                        sens_1=11;
                                        sens_2=13;
                                        if(sens.values[sens_1]>sens.values[sens_2])
                                                {
                                                        mov.left=1;
                                                        mov.right=-1;
                                                }
                                                else
                                                        {
                                                                mov.left=-1;
                                                                mov.right=1;
                                                        };
                                }; /* end of BACK */
                        if (side==LEFT)
                                {
                                        sens_1=17;
                                        sens_2=19;
                                        if(sens.values[sens_1]>sens.values[sens_2])
                                                {
                                                        mov.left=1;
                                                        mov.right=-1;
                                                }
                                                else
                                                        {
                                                                mov.left=-1;
                                                                mov.right=1;
                                                        };
                                }; /* end of LEFT */
                        if(sens.values[sens_1]>sens.values[sens_2])
                                {
                                        dif=(sens.values[sens_1] - sens.values[sens_2]);
                                }
                                else
                                        {
                                                dif=(sens.values[sens_2] - sens.values[sens_1]);
                                        };
                        if(dif<0)
                                dif=-dif;
                        if(dif<f_parallel.erro)
                                {
                                        mov.left=0;   /* stop */
                                        mov.right=0;
                                        f_parallel.state++;
                                };
                }; /* end of state 1 */

        if(f_parallel.state == 2)
                {
                        printf("\n Fim de calibracao \n");
                        f_parallel.state=0;
                        mission_state++;
                }; /* end of state 2 */

return(f_parallel.state); /* return the state of the function to main program */
}; /* end of parallel */
/*****************************************************************************/
/* movimento que percorre a direccao actual ate a uma distancia x com uma velocidade v */

int go(int dist,int v_in)
{
        int aux=0;


        dist=dist*100;


        if(v_in>=100)           /* limit of v_in */
           v_in=100;
        if(v_in<=-100)          /* limit of v_in */
           v_in=-100;
        if(dist<0)
                {
                     dist=-dist;
                     printf("\nWarning (bad argument in go (dist<0) ");
                 };
```

```
        if(f_go.state==0)
                {
                        f_go.old_x=pos.x;
                        f_go.old_y=pos.y;
                        f_go.old_theta=pos.theta;
                        sendf("ODOS C=%ld,%ld,%ld",0,0,0);   /* reset dos encoders */
                        pos.x=0;
                        pos.y=0;
                        pos.theta=0;
                        f_go.erro=v_in/10;
                        f_go.state=1;
                };                                           /* end of state 0 */
        if(f_go.state==1)
                {
                        if(dist>20000)
                                aux=3000;
                                        else
                                                aux=2000;
                        if(v_in>0)
                                {
                                        f_go.v=v_in;
                                        if( pos.x < 200)
                                            f_go.v/=3;
                                        if( pos.x < 50)
                                            f_go.v/=5;
                                        if(dist-pos.x<aux)
                                            f_go.v/=2;
                                        if(dist-pos.x<(aux/3))
                                            f_go.v/=10;
                                        if(f_go.v<3)
                                            f_go.v=3;
                                };
                        if(v_in<0)
                                {
                                        f_go.v=-v_in;
                                        if(-pos.x<200)
                                            f_go.v/=3;
                                        if(-pos.x<50)
                                            f_go.v/=5;
                                        if(dist+pos.x < aux)
                                            f_go.v/=2;
                                        if(dist+pos.x< (aux/3))
                                            f_go.v/=10;
                                        if(f_go.v<3)
                                            f_go.v=3;
                                        f_go.v= -f_go.v;
                                };
                        mov.left=f_go.v;
                        mov.right=f_go.v;

                        if(v_in>0)                           /* go front */
                           if(pos.x>= dist)
                                {
                                    mov.left=0;              /* stop */
                                    mov.right=0;
                                    f_go.v=0;
                                    sendf("ODOS C=%ld,%ld,%ld",pos.x+f_go.old_x,pos.y+f_go.old_y,pos.theta+f_go.old_theta);
                                    f_go.state=0;
                                    mission_state++;
                                };
                        if(v_in<0)                           /* go back */
                            if(pos.x<= -dist)
                                {
                                    mov.left=0;              /* stop */
                                    mov.right=0;
                                    f_go.v=0;
                                    sendf("ODOS C=%ld,%ld,%ld",pos.x+f_go.old_x,pos.y+f_go.old_y,pos.theta+f_go.old_theta);    ...

                                    f_go.state=0;
                                    mission_state++;
                                };
                }; /* end of state 1 */
        return(f_go.state);
}; /* end of go function */

/*************************************************************************/
/* movimento rotativo com angulo theta e velocidade v                    */

int roll(long int theta_in, int velocity_in)
{
        f_roll.erro=5;

        if(velocity_in>60)                   /* limit of velocity_in */
           velocity_in=60;
        if(velocity_in<-60)
           velocity_in=-60;

        if(theta_in>0)
                theta_in=theta_in-f_roll.erro;
        if(theta_in<0)
                theta_in=theta_in+f_roll.erro;
        f_roll.velocity=velocity_in;
        if(theta_in==180)
                theta_in=theta_in-1;
        if(theta_in==-180)
                theta_in=theta_in+1;

        if((theta_in==-180) || (theta_in==-180))
                velocity_in=10;

        switch (f_roll.state)
        {
                case 0:         f_roll.old_x=pos.x;
                                f_roll.old_y=pos.y;
                                f_roll.old_theta=pos.theta;
                                sendf("ODOS C=%ld,%ld,%ld",0,0,0);
                                pos.x=0;
                                pos.y=0;
```

```
                                pos.theta=0;                                                                    956
                                f_roll.state=1;
                                break;
                case 1:         if(theta_in < 0)    /* turn right */
                                        {
                                              if(-theta_in+pos.theta<30)
                                                  f_roll.velocity/=2;
                                              if(-theta_in+pos.theta<10)
                                                  f_roll.velocity/=10;
                                              if(f_roll.velocity<=3)
                                                  f_roll.velocity=3;
                                              mov.left=f_roll.velocity;
                                              mov.right=-f_roll.velocity;
                                        };
                                if(theta_in > 0)    /* turn  left */
                                        {
                                                if (theta_in-pos.theta<=30)
                                                    f_roll.velocity/=2;
                                                if (theta_in-pos.theta<=10)
                                                    f_roll.velocity/=10;
                                                if(f_roll.velocity<=3)
                                                    f_roll.velocity=3;

                                              mov.left=-f_roll.velocity;
                                              mov.right=f_roll.velocity;
                                        };
                                if((theta_in>=pos.theta)&&(theta_in< 0))
                                        {
                                                mov.left=0;                                     /* stop right */
                                                mov.right=0;
                                                theta_in=0;
                                                f_roll.velocity=0;
                                                sendf("ODOS C=%ld,%ld,%ld"                                      ...
,pos.x+f_roll.old_x,pos.y+f_roll.old_y,f_roll.old_theta+pos.theta);
                                                f_roll.old_theta=0;
                                                f_roll.state=0;
                                                mission_state++;
                                        };
                                if((theta_in<=pos.theta)&&(theta_in> 0))
                                        {
                                                mov.left=0;                                     /* stop left */
                                                mov.right=0;
                                                theta_in=0;
                                                f_roll.velocity=0;
                                                sendf("ODOS C=%ld,%ld,%ld"                                      ...
,pos.x+f_roll.old_x,pos.y+f_roll.old_y,f_roll.old_theta+pos.theta);
                                                f_roll.old_theta=0;
                                                mission_state++;
                                                f_roll.state=0;
                                        };
                        break;  /* end of case1 */
                }; /*end of case */
return(f_roll.state);
} /* end of roll function */

/*********************************************************************/
/* movimento cicular com raio r, angulo theta e velocidade v         */

int circ(float raio,long int theta_in, int v_in)
{
        float d=0.62;
        f_circ.erro=3;

        raio=raio*100;

        if(theta_in>0)
                theta_in=theta_in-f_circ.erro;
        if(theta_in<0)
                theta_in=theta_in+f_circ.erro;
        raio/=10000;
        switch (f_circ.state)
        {
                case 0:
                                f_circ.old_x=pos.x;
                                f_circ.old_y=pos.y;
                                f_circ.old_theta=pos.theta;
                                sendf("ODOS C=%ld,%ld,%ld",0,0,0);
                                pos.theta=0;
                                f_circ.state=1;
                                break;

                case 1:         if(theta_in>0)
                                        {
                                            if(pos.theta<5)
                                                v_in/=2;
                                            if((theta_in-pos.theta)<5)
                                                v_in/=2;
                                        };
                                if(theta_in<0)
                                        {
                                            if(-pos.theta<5)
                                                v_in/=2;
                                            if(-theta_in+pos.theta<5)
                                                v_in/=2;
                                        };
                                f_circ.vr=((2*v_in)-((v_in*(raio-(d/2)))/raio));  /* Calculo das velocidades */
                                f_circ.vl=((v_in*(raio-(d/2)))/raio);

                                if(theta_in>0)
                                        {
                                                mov.left=f_circ.vl/1;
                                                mov.right=f_circ.vr/1;
                                        };
                                if(theta_in<0)
                                        {
                                                mov.left=f_circ.vr/1;
                                                mov.right=f_circ.vl/1;
                                        };
```

```
                                if((theta_in<=pos.theta)&&(theta_in> 0))                                        1062
                                        {
                                                mov.left=0;                                     /* stop left */
                                                mov.right=0;
                                                theta_in=0;
                                                f_circ.vr=0;
                                                f_circ.vl=0;
                                                sendf("ODOS C=%ld,%ld,%ld"                                      ...
,f_circ.old_x+pos.x,f_circ.old_y+pos.y,f_circ.old_theta+pos.theta);
                                                f_circ.old_x=0;
                                                f_circ.old_y=0;
                                                f_circ.old_theta=0;
                                                mission_state++;
                                                f_circ.state=0;
                                        };
                                if((theta_in>=pos.theta)&&(theta_in< 0))
                                        {
                                                mov.left=0;                                     /* stop right */
                                                mov.right=0;
                                                theta_in=0;
                                                f_circ.vr=0;
                                                f_circ.vl=0;
                                                sendf("ODOS C=%ld,%ld,%ld"                                      ...
,f_circ.old_x+pos.x,f_circ.old_y+pos.y,f_circ.old_theta+pos.theta);
                                                f_circ.old_x=0;
                                                f_circ.old_y=0;
                                                f_circ.old_theta=0;
                                                mission_state++;
                                                f_circ.state=0;
                                        };
                        break;  /* end of case1 */
                }; /*end of case */
return(f_circ.state);
} /* end of circ function */
/*************************************************************************/
/* movimento de concordancia com raio r, angulo theta e velocidade v     */

int curve(float l,long int theta_in, int v_in)
{
        float raio=0,l1=0;
        float d=0.62;
        float theta1=0;

        l=l*100;

        if (l<1)
                l=1;
        f_curve.erro=2;
        if(v_in>=20)
                f_curve.erro=4;
        if(theta_in>0)
                {
                        theta_in=theta_in-f_curve.erro;
                        theta1=theta_in;
                };
        if(theta_in<0)
                {
                        theta_in=theta_in+f_curve.erro;
                        theta1=-theta_in;
                };
        l1=l/10000;                                             /* conversao para metros   */
        theta1=((theta1*3.141592654)/180);                      /* graus para radianos     */
        raio=(((l1*cos(theta1))+l1)/sin(theta1));               /* Calculo do raio de curvatura */

        switch (f_curve.state)
        {
                case 0:
                                f_curve.old_x=pos.x;
                                f_curve.old_y=pos.y;
                                f_curve.old_theta=pos.theta;
                                sendf("ODOS C=%ld,%ld,%ld",0,0,0);
                                pos.theta=0;
                                f_curve.state=1;
                                break;

                case 1:         if(theta_in>0)
                                        {
                                            if(pos.theta<5)
                                                v_in/=2;
                                            if((theta_in-pos.theta)<5)
                                                v_in/=2;
                                        };
                                if(theta_in<0)
                                        {
                                            if(-pos.theta<5)
                                                v_in/=2;
                                            if(-theta_in+pos.theta<5)
                                                v_in/=2;
                                        };
                                f_curve.vr=((2*v_in)-((v_in*(raio-(d/2)))/raio));   /* Calculo das velocidades */
                                f_curve.vl=((v_in*(raio-(d/2)))/raio);

                                if(theta_in>0)
                                        {
                                                mov.left=f_curve.vl/1;
                                                mov.right=f_curve.vr/1;
                                        };
                                if(theta_in<0)
                                        {
                                                mov.left=f_curve.vr/1;
                                                mov.right=f_curve.vl/1;
                                        };
                                if((theta_in<=pos.theta)&&(theta_in> 0))
                                        {
                                                mov.left=0;                                     /* stop left */
                                                mov.right=0;
                                                theta_in=0;
                                                f_curve.vr=0;
```

```
                                        f_curve.vl=0;                                                   1168
                                        sendf("ODOS C=%ld,%ld,%ld"                                      ...
,f_curve.old_x+pos.x,f_curve.old_y+pos.y,f_curve.old_theta+pos.theta);
                                        f_curve.old_x=0;
                                        f_curve.old_y=0;
                                        f_curve.old_theta=0;
                                        mission_state++;
                                        f_curve.state=0;
                                    };
                                if((theta_in>=pos.theta)&&(theta_in< 0))
                                    {
                                        mov.left=0;                         /* stop right */
                                        mov.right=0;
                                        theta_in=0;
                                        f_curve.vr=0;
                                        f_curve.vl=0;
                                        sendf("ODOS C=%ld,%ld,%ld"                                      ...
,f_curve.old_x+pos.x,f_curve.old_y+pos.y,f_curve.old_theta+pos.theta);
                                        f_curve.old_x=0;
                                        f_curve.old_y=0;
                                        f_curve.old_theta=0;
                                        mission_state++;
                                        f_curve.state=0;
                                    };
                            break;    /* end of case1 */
                        }; /*end of case */
return(f_curve.state);
} /* end of curve function */
```

```
/* S.N.A.N. para RobuterIII

  Anabela Duarte  &  Paulo Peixoto
**********************************************************************/

#define ERROR       -1
#define OK           1

#define MAX_LENGTH 500          /* Maximum message length */
#define MAX_PARAMETERS 10       /* Maximum number of task parameters */

#define SEPARATOR1   32
#define SEPARATOR2   44


#define TERMINATOR   13
#define TERMINATOR2   10
#define MAX_TASKS 100

/**********************************************************************/

#define MSGC_DATAQUERY  'B'
        #define IN_POST       'O'
        #define IN_US         'U'
        #define IN_POSTUS     'V'
        #define IN_CLOCK      'C'
        #define IN_USTOUT     'T'
        #define IN_PTU_US     'P'
        #define IN_PTU_FUS    'F'
        #define IN_PTU_DUS    'D'
/**********************************************************************/

#define MSGC_UNIDIRCOM  'A'
        #define IN_VELOCITY   'M'
        #define IN_POSITION   'W'
        #define IN_ODOM       'C'
        #define IN_SERV       'S'
        #define IN_ALBATROS   'Y'
        #define IN_KILL       'X'
        #define IN_DOOR       'd'
        #define IN_DOOR_OFF   'f'
        #define IN_PTU_ON     'p'
        #define IN_PTU_OFF    'g'
        #define IN_AUTO_ON    'a'
        #define IN_AUTO_OFF   'm'
        #define IN_DEM_OFF    'o'
        #define DIST_DEMO     'D'
/**********************************************************************/

#define MSGC_USACTION   'U'
        #define IN_USTABLE    'T'
        #define IN_NDELAY     'D'
        #define IN_GNDELAY    'N'
        #define IN_STAT       'S'
        #define IN_SUSTOUT    't'
        #define IN_TOGSENS    'u'
/**********************************************************************/
#define MSGC_MISSION_COM      'C'       /* Categoty mission command      */

        #define START         'A'       /* Start the mission processing    */
        #define STOP          'S'       /* Stop the mission processing            */
        #define PAUSE         'P'       /* Stop and wait                  */
        #define CLEAR         'E'       /* Clear the thask mission array   */
        #define INSERT        'I'       /* Insert a task in mission array  */
        #define REPLACE       'R'       /* Replace a task in mission array  */
        #define DELETE        'D'       /* Delete a task in mission array   */
        #define LIST          'L'       /* Give the list of tasks to execute */
        #define TASK          'T'       /* Manual task processing          */

/**********************************************************************/
/*                                    Mission type (parameter 0 )      */
        #define T_GO            1
        #define T_ROLL          2
        #define T_CURVE         3
        #define T_CIRC          4
        #define T_PARALLEL      5
        #define T_GO_UNTIL      6
        #define T_CROSS_DOOR    7
        #define T_GO_PARALELL   8
        #define T_GO_AT         9
        #define T_ACTIVE_US     10

/**********************************************************************/

typedef struct {
        int state;                                      /* State of task      */
        int next;                                       /* Next task to execute */
        long int parameter[MAX_PARAMETERS];             /* Parameter of task          */

} S_MISSION_TASK;

/**********************************************************************/

typedef struct {
  char buff_in[MAX_LENGTH];
  char buff_out[MAX_LENGTH];
  int desc, send;
} SERIAL;

/**********************************************************************/

void decode(void);              /* Performs all actions related with received message */
int init_serial_spy(void);
void close_serial_spy(void);
void serial_spy(void);          /* Process that "spys" the serial line */
void init_mission_array(void);
void print_task(int i);
int detect_first_empty(void);
```

```
int detect_previous(int i);
void actualize_mission_manager(void);
```

```
/* S.N.A.N. para RobuterIII

  Anabela Duarte  &  Paulo Peixoto

*****************************************************************/

#include <libcstd.h>
#include <libport.h>
#include <libcalbl.h>
#include <libcpro.h>

#include "generic.h"
#include "us.h"
#include "executor.h"
#include "serial.h"
#include "emerg.h"
#include "headers.h"

#include "ptu.h"
#include "doors.h"


#include "detect.h"
#include "mission_executor.h"

#define PORT "SLB"

extern int kill;
extern US_DATA sens;
extern POSTURE pos;
extern MOVES mov;
extern EMERGENCY emerg;


extern long pt_values[MAX_PTU_SENS];
extern long fvalues[MAX_PTU_SENS];
extern long dvalues[MAX_PTU_SENS];
extern int ptu_on;
extern int door;
extern int auto_detect;


static SERIAL serial;


extern S_ROLL f_roll;
extern S_GO f_go;
extern S_CIRC f_circ;
extern S_CURVE f_curve;
extern S_PARALLEL f_parallel;
extern S_GOU f_gou;
extern S_CROSS_DOOR f_cross_door;
extern S_GO_PARALLEL f_go_parallel;
extern S_GO_AT f_go_at;
extern S_MANUAL_TYPE f_manual_type;

int free_cell;
int tail;

/*****************************************************************************/

S_MISSION_TASK mission_array[MAX_TASKS];
int mission_manager[MAX_TASKS];

int mission_state;
int position_array;
int start_mission;

/*****************************************************************************/

void decode(void)
{

        int i;
        int insert_next;
        int parameter0;
        long int parameter1;
        long int parameter2;
        long int parameter3;
        long int parameter4;
        long int parameter5;
        char category, type, aux[50];
        char param[5][30], *token, *tmp;
        int p, n_param;


        if (strlen(serial.buff_in)<=2)
                {
                        strcpy(serial.buff_out,"error\r");       /* Se existem menos de dois caracteres */
                        serial.send=TRUE;                        /*  envia msg de erro */
                        return;
                }

        token=serial.buff_in;

        category=*token;                                         /* Retira a CATEGORIA */
        token++;
        type=*token;                                             /* Retira o TIPO */
        token++;

        p=0;                                                     /* Retirar parametros, */
        while (*token!=TERMINATOR)
                {
                        tmp=aux;                                 /*   se existirem.   */
                        do
                                {
                                        *tmp=*token;
                                        tmp++;
```

```
                                token++;                                                    109
                        } while
( (*token!=TERMINATOR) && ( ( (*token!=SEPARATOR1) && (*token!=SEPARATOR2) ) || (type==IN_ALBATROS) ) );
                        *tmp='\0';
                        strcpy(param[p],aux);
                        if(*token!=TERMINATOR)
                                token++;
                        p++;
                        n_param=p;
                }

                parameter0=atoi(param[0]);
                parameter1=atoi(param[1]);
                parameter2=atoi(param[2]);
                parameter3=atoi(param[3]);
                parameter4=atoi(param[4]);
                parameter5=atoi(param[5]);

                free_cell=detect_first_empty();

                if(free_cell>MAX_TASKS)
                        free_cell=0;
/*****************************************************************************/
                switch(category)
                   {
                        case TASK:
                                switch(parameter0)
                                        {
                                                case T_GO:
                                                        mission_array[free_cell].state=1;
                                                        mission_array[free_cell].parameter[0]=T_GO;
                                                        mission_array[free_cell].parameter[1]=parameter1;
                                                        mission_array[free_cell].parameter[2]=parameter2;
                                                        mission_array[free_cell].next=0;
                                                        mission_array[tail].next=free_cell;
                                                        tail=free_cell;
                                                        break;
                                                case T_ROLL:
                                                        mission_array[free_cell].state=1;
                                                        mission_array[free_cell].parameter[0]=T_ROLL;
                                                        mission_array[free_cell].parameter[1]=parameter1;
                                                        mission_array[free_cell].parameter[2]=parameter2;
                                                        mission_array[free_cell].next=0;
                                                        mission_array[tail].next=free_cell;
                                                        tail=free_cell;                                                    ...

                                                        break;
                                                case T_CURVE:
                                                        mission_array[free_cell].state=1;
                                                        mission_array[free_cell].parameter[0]=T_CURVE;
                                                        mission_array[free_cell].parameter[1]=parameter1;
                                                        mission_array[free_cell].parameter[2]=parameter2;
                                                        mission_array[free_cell].parameter[3]=parameter3;
                                                        mission_array[free_cell].next=0;
                                                        mission_array[tail].next=free_cell;
                                                        tail=free_cell;
                                                        break;
                                                case T_CIRC:
                                                        mission_array[free_cell].state=1;
                                                        mission_array[free_cell].parameter[0]=T_CIRC;
                                                        mission_array[free_cell].parameter[1]=parameter1;
                                                        mission_array[free_cell].parameter[2]=parameter2;
                                                        mission_array[free_cell].parameter[3]=parameter3;
                                                        mission_array[free_cell].next=0;
                                                        mission_array[tail].next=free_cell;
                                                        tail=free_cell;
                                                        break;
                                                case T_PARALLEL:
                                                        mission_array[free_cell].state=1;
                                                        mission_array[free_cell].parameter[0]=T_PARALLEL;
                                                        mission_array[free_cell].parameter[1]=parameter1;
                                                        mission_array[free_cell].next=0;
                                                        mission_array[tail].next=free_cell;
                                                        tail=free_cell;
                                                        break;
                                                case T_GO_UNTIL:
                                                        mission_array[free_cell].state=1;
                                                        mission_array[free_cell].parameter[0]=T_GO_UNTIL;
                                                        mission_array[free_cell].parameter[1]=parameter1;
                                                        mission_array[free_cell].parameter[2]=parameter2;
                                                        mission_array[free_cell].parameter[3]=parameter3;
                                                        mission_array[free_cell].next=0;
                                                        mission_array[tail].next=free_cell;
                                                        tail=free_cell;
                                                        break;
                                                case T_CROSS_DOOR:
                                                        mission_array[free_cell].state=1;
                                                        mission_array[free_cell].parameter[0]=T_CROSS_DOOR;
                                                        mission_array[free_cell].next=0;
                                                        mission_array[tail].next=free_cell;
                                                        tail=free_cell;
                                                        break;
                                                case T_GO_PARALELL:
                                                        mission_array[free_cell].state=1;
                                                        mission_array[free_cell].parameter[0]=T_GO_PARALELL;
                                                        mission_array[free_cell].parameter[1]=parameter1;
                                                        mission_array[free_cell].parameter[2]=parameter2;
                                                        mission_array[free_cell].parameter[3]=parameter3;
                                                        mission_array[free_cell].next=0;
                                                        mission_array[tail].next=free_cell;
                                                        tail=free_cell;
                                                        break;
                                                case T_GO_AT:
                                                        mission_array[free_cell].state=1;
                                                        mission_array[free_cell].parameter[0]=T_GO_AT;
                                                        mission_array[free_cell].parameter[1]=parameter1;
                                                        mission_array[free_cell].parameter[2]=parameter2;
                                                        mission_array[free_cell].parameter[3]=parameter3;
```

```
                                                mission_array[free_cell].parameter[4]=parameter4;
                                                mission_array[free_cell].next=0;
                                                mission_array[tail].next=free_cell;
                                                tail=free_cell;
                                                break;
                                        case T_ACTIVE_US:
                                                mission_array[free_cell].state=1;
                                                mission_array[free_cell].parameter[0]=T_ACTIVE_US;
                                                mission_array[free_cell].parameter[1]=parameter1;
                                                mission_array[free_cell].parameter[2]=parameter2;
                                                mission_array[free_cell].parameter[3]=parameter3;
                                                mission_array[free_cell].parameter[4]=parameter4;
                                                mission_array[free_cell].next=0;
                                                mission_array[tail].next=free_cell;
                                                tail=free_cell;
                                                break;
                                        }; /*end type of task */
                                break;
/*****************************************************************************/
                        case MSGC_MISSION_COM:
                                switch(type)
                                   {
                                        case START:
                                                f_manual_type.state=0;
                                                start_mission=1;

                                                printf("\nStart Mission\n");
                                                strcpy(serial.buff_out,"ack\r");
                                                serial.send=TRUE;
                                                break;

                                        case STOP:
                                                mov.left=0;
                                                mov.right=0;
                                                f_manual_type.state=0;
                                                start_mission=0;
                                                reset_tasks_states();

                                                strcpy(serial.buff_out,"ack\r");
                                                serial.send=TRUE;
                                                break;

                                        case PAUSE:
                                                mov.left=0;
                                                mov.right=0;
                                                if((f_manual_type.state==1) || ( start_mission ==1))
                                                        {
                                                                f_manual_type.state=0;
                                                                start_mission=0;
                                                        }
                                                        else
                                                                {
                                                                        f_manual_type.state=1;
                                                                        start_mission=1;
                                                                };

                                                strcpy(serial.buff_out,"ack\r");
                                                serial.send=TRUE;
                                                break;

                                        case CLEAR:
                                                mov.left=0;
                                                mov.right=0;
                                                emerg.rtm=TRUE;
                                                init_mission_array();

                                                f_manual_type.state=0;
                                                start_mission=0;
                                                mission_state=0;
                                                position_array=0;
                                                tail=0;

                                                strcpy(serial.buff_out,"ack\r");
                                                serial.send=TRUE;
                                                break;

                                        case INSERT:
                                                if(parameter0 != -1)
                                                        if(mission_array[parameter0].state != 1)
                                                                {
                                                                        printf("\nPermission denied");
                                                                        printf("\nInvalid task number\n");
                                                                        break;
                                                                };

                                                if((parameter0 == position_array) && (start_mission==1))
                                                        {
                                                                printf("\npermission denied");
                                                                printf("\nthis tasks is in process\n");
                                                                break;
                                                        };
                                                free_cell=detect_first_empty();
                                                if(parameter0 == -1)
                                                        {
                                                                position_array=free_cell;
                                                                parameter0=0;
                                                        };
                                                if(parameter0 == tail)
                                                        {
                                                                insert_next=0;
                                                                tail=free_cell;
                                                        }
                                                        else
                                                                {
                                                                        insert_next=mission_array[parameter0].next;

                                                                };
                                                switch(parameter1)
```

```
{                                                                                    321
        case T_GO:
                mission_array[free_cell].state=1;                                    ...

                mission_array[free_cell].parameter[0]=T_GO;
mission_array[free_cell].parameter[1]=parameter2;                                    ...
mission_array[free_cell].parameter[2]=parameter3;                                    ...

                mission_array[free_cell].next=insert_next;
                mission_array[parameter0].next=free_cell;
                break;

        case T_ROLL:
                mission_array[free_cell].state=1;                                    ...
mission_array[free_cell].parameter[0]=T_ROLL;                                        ...
mission_array[free_cell].parameter[1]=parameter2;                                    ...
mission_array[free_cell].parameter[2]=parameter3;

                mission_array[free_cell].next=insert_next;
                mission_array[parameter0].next=free_cell;                            ...

                break;

        case T_CURVE:
                mission_array[free_cell].state=1;                                    ...
mission_array[free_cell].parameter[0]=T_CURVE;                                       ...
mission_array[free_cell].parameter[1]=parameter2;                                    ...
mission_array[free_cell].parameter[2]=parameter3;                                    ...
mission_array[free_cell].parameter[3]=parameter4;

                mission_array[free_cell].next=insert_next;
                mission_array[parameter0].next=free_cell;
                break;

        case T_CIRC:
                mission_array[free_cell].state=1;
mission_array[free_cell].parameter[0]=T_CIRC;                                        ...
mission_array[free_cell].parameter[1]=parameter2;                                    ...
mission_array[free_cell].parameter[2]=parameter3;                                    ...
mission_array[free_cell].parameter[3]=parameter4;

                mission_array[free_cell].next=insert_next;
                mission_array[parameter0].next=free_cell;
                break;

        case T_PARALLEL:
                mission_array[free_cell].state=1;
mission_array[free_cell].parameter[0]=T_PARALLEL;                                    ...
mission_array[free_cell].parameter[1]=parameter2;                                    ...

                mission_array[free_cell].next=insert_next;
                mission_array[parameter0].next=free_cell;
                break;
        case T_GO_UNTIL:
                mission_array[free_cell].state=1;
mission_array[free_cell].parameter[0]=T_GO_UNTIL;                                    ...
mission_array[free_cell].parameter[1]=parameter2;                                    ...
mission_array[free_cell].parameter[2]=parameter3;                                    ...
mission_array[free_cell].parameter[3]=parameter4;                                    ...

                mission_array[free_cell].next=insert_next;
                mission_array[parameter0].next=free_cell;
                break;

        case T_CROSS_DOOR:
                mission_array[free_cell].state=1;
mission_array[free_cell].parameter[0]=T_CROSS_DOOR;                                  ...

                mission_array[free_cell].next=insert_next;
                mission_array[parameter0].next=free_cell;
                break;

        case T_GO_PARALELL:
                mission_array[free_cell].state=1;
mission_array[free_cell].parameter[0]=T_GO_PARALELL;                                 ...
mission_array[free_cell].parameter[1]=parameter2;                                    ...
mission_array[free_cell].parameter[2]=parameter3;                                    ...
mission_array[free_cell].parameter[3]=parameter4;                                    ...

                mission_array[free_cell].next=insert_next;
                mission_array[parameter0].next=free_cell;
                break;

        case T_GO_AT:
                mission_array[free_cell].state=1;
mission_array[free_cell].parameter[0]=T_GO_AT;                                       ...
mission_array[free_cell].parameter[1]=parameter2;                                    ...
                                                                                     ...
```

```
mission_array[free_cell].parameter[2]=parameter3;                                   398
mission_array[free_cell].parameter[3]=parameter4;                                   ...
mission_array[free_cell].parameter[4]=parameter5;                                   ...
                                                        mission_array[free_cell].next=insert_next;
                                                        mission_array[parameter0].next=free_cell;
                                                        break;

                                                case T_ACTIVE_US:
                                                        mission_array[free_cell].state=1;           ...
                                                                                                    ...
mission_array[free_cell].parameter[0]=T_ACTIVE_US;                                  ...
mission_array[free_cell].parameter[1]=parameter2;                                   ...
mission_array[free_cell].parameter[2]=parameter3;                                   ...
mission_array[free_cell].parameter[3]=parameter4;                                   ...
mission_array[free_cell].parameter[4]=parameter5;
                                                        mission_array[free_cell].next=insert_next;
                                                        mission_array[parameter0].next=free_cell;
                                                        break;
                                                default:
                                                        printf("\nInvalid task type\n");
                                                        printf(                             ...
"Please, REPLACE with valid arguments !\n");
                                                        break;

                                                }; /*end type of task(INSERT) */
                                        strcpy(serial.buff_out, "ack\r");
                                        serial.send=TRUE;
                                        break;

                                case REPLACE:
                                        if(mission_array[parameter0].state != 1)
                                                {
                                                        printf("\n\nPermission denied");
                                                        printf("\nInvalid task number\n");
                                                        break;
                                                };
                                        mission_array[parameter0].parameter[0]=0;
                                        mission_array[parameter0].parameter[1]=0;
                                        mission_array[parameter0].parameter[2]=0;
                                        mission_array[parameter0].parameter[3]=0;
                                        mission_array[parameter0].parameter[4]=0;
                                        mission_array[parameter0].parameter[5]=0;

                                        switch(parameter1)
                                                {
                                                        case T_GO:
                                                                mission_array[parameter0].state=1;  ...
                                                                                                    ...
mission_array[parameter0].parameter[0]=T_GO;                                        ...
mission_array[parameter0].parameter[1]=parameter2;                                  ...
mission_array[parameter0].parameter[2]=parameter3;                                  ...
                                                                break;
                                                        case T_ROLL:
                                                                mission_array[parameter0].state=1;  ...
                                                                                                    ...
mission_array[parameter0].parameter[0]=T_ROLL;                                      ...
mission_array[parameter0].parameter[1]=parameter2;                                  ...
mission_array[parameter0].parameter[2]=parameter3;                                  ...
                                                                break;
                                                        case T_CURVE:
                                                                mission_array[parameter0].state=1;  ...
                                                                                                    ...
mission_array[parameter0].parameter[0]=T_CURVE;                                     ...
mission_array[parameter0].parameter[1]=parameter2;                                  ...
mission_array[parameter0].parameter[2]=parameter3;                                  ...
mission_array[parameter0].parameter[3]=parameter4;                                  ...
                                                                break;
                                                        case T_CIRC:
                                                                mission_array[parameter0].state=1;
                                                                                                    ...
mission_array[parameter0].parameter[0]=T_CIRC;                                      ...
mission_array[parameter0].parameter[1]=parameter2;                                  ...
mission_array[parameter0].parameter[2]=parameter3;                                  ...
mission_array[parameter0].parameter[3]=parameter4;                                  ...
                                                                break;
                                                        case T_PARALLEL:
                                                                mission_array[parameter0].state=1;
                                                                                                    ...
mission_array[parameter0].parameter[0]=T_PARALLEL;                                  ...
mission_array[parameter0].parameter[1]=parameter2;
                                                                break;
                                                        case T_GO_UNTIL:
                                                                mission_array[parameter0].state=1;
                                                                                                    ...
mission_array[parameter0].parameter[0]=T_GO_UNTIL;                                  ...
mission_array[parameter0].parameter[1]=parameter2;
```

```
…
mission_array[parameter0].parameter[3]=parameter4;
                                                        break;
                                                case T_CROSS_DOOR:
                                                        mission_array[parameter0].state=1;
mission_array[parameter0].parameter[0]=T_CROSS_DOOR;
                                                        break;
                                                case T_GO_PARALELL:
                                                        mission_array[parameter0].state=1;
mission_array[parameter0].parameter[0]=T_GO_PARALELL;
mission_array[parameter0].parameter[1]=parameter2;
mission_array[parameter0].parameter[2]=parameter3;
mission_array[parameter0].parameter[3]=parameter4;
                                                        break;
                                                case T_GO_AT:
                                                        mission_array[parameter0].state=1;
mission_array[parameter0].parameter[0]=T_GO_AT;
mission_array[parameter0].parameter[1]=parameter2;
mission_array[parameter0].parameter[2]=parameter3;
mission_array[parameter0].parameter[3]=parameter4;
mission_array[parameter0].parameter[4]=parameter5;
                                                        break;
                                                case T_ACTIVE_US:
                                                        mission_array[parameter0].state=1;
mission_array[parameter0].parameter[0]=T_ACTIVE_US;
mission_array[parameter0].parameter[1]=parameter2;
mission_array[parameter0].parameter[2]=parameter3;
mission_array[parameter0].parameter[3]=parameter4;
mission_array[parameter0].parameter[4]=parameter5;
                                                        break;
                                                default:
                                                        printf("\nInvalid task type\n");
                                                        printf(
"Please, replace with valid arguments !\n");
                                                        break;
                                                }; /*end type of task(REPLACE) */

                                        strcpy(serial.buff_out,"ack\r");
                                        serial.send=TRUE;
                                        break;

                                case DELETE:
                                        if(mission_array[parameter0].state != 1)
                                                {
                                                        printf("\n\nPermission denied");
                                                        printf("\nInvalid task number\n");
                                                        break;
                                                };

                                        if(parameter0 == position_array)
                                                {
                                                        if(start_mission==1)
                                                                {
                                                                        printf(
"\n\npermission denied");
                                                                        printf(
"\nthis tasks is in process\n");
                                                                }
                                                        else
                                                                {
position_array=mission_array[parameter0].next;
mission_array[parameter0].next=0;
mission_array[parameter0].state=0;
mission_array[parameter0].next=0;
mission_array[parameter0].parameter[0]=0;
mission_array[parameter0].parameter[1]=0;
mission_array[parameter0].parameter[2]=0;
mission_array[parameter0].parameter[3]=0;
mission_array[parameter0].parameter[4]=0;
mission_array[parameter0].parameter[5]=0;
                                                                        };
                                                }
                                        else
                                                {
                                                        if(parameter0==tail)
                                                                {
tail=detect_previous(parameter0);
```

```
…
mission_array[parameter0].state=0;
mission_array[parameter0].next=0;
mission_array[parameter0].parameter[0]=0;
mission_array[parameter0].parameter[1]=0;
mission_array[parameter0].parameter[2]=0;
mission_array[parameter0].parameter[3]=0;
mission_array[parameter0].parameter[4]=0;
mission_array[parameter0].parameter[5]=0;
                                                                    }
                                                                    else
                                                                        {
mission_array[detect_previous(parameter0)].next=mission_array[parameter0].next;
mission_array[parameter0].state=0;
mission_array[parameter0].next=0;
mission_array[parameter0].parameter[0]=0;
mission_array[parameter0].parameter[1]=0;
mission_array[parameter0].parameter[2]=0;
mission_array[parameter0].parameter[3]=0;
mission_array[parameter0].parameter[4]=0;
mission_array[parameter0].parameter[5]=0;
                                                                        };
                                                    };
                                        strcpy(serial.buff_out,"ack\r");
                                        serial.send=TRUE;
                                        break;

                                case LIST:
                                        printf("\n*************************************\n");
                                        printf("\nList of tasks to execute\n");
                                        printf("\n num.  Ref.       Task\n");

                                        actualize_mission_manager();

                                        if(mission_array[mission_manager[0]].state==1)
                                                {
                                                        printf("  %d   ",0);
                                                        print_task(0);
                                                };

                                        for(i=1;i<= MAX_TASKS ;i++)
                                                if(mission_manager[i] != 0)
                                                        {
                                                                if(i<10)
                                                                        printf(" ");
                                                                printf(" %d  ",i);
                                                                if(mission_manager[i]<10)
                                                                        printf(" ");
                                                                print_task(i);
                                                        };

                                        strcpy(serial.buff_out,"ack\r");
                                        serial.send=TRUE;
                                        break;

                                case TASK:
                                        switch(parameter0)
                                                {
                                                        case T_GO:
                                                                if(f_manual_type.state==0)
                                                                        {
                                                                                f_manual_type.state=1;
                                                                                f_manual_type.type=T_GO;
f_manual_type.parameter1=parameter1;
f_manual_type.parameter2=parameter2;
                                                                                strcpy(serial.buff_out,
"ack\r");
                                                                                serial.send=TRUE;
                                                                        }
                                                                        else
                                                                        {
                                                                             printf(
"\nPermission denied\n");
                                                                        };
                                                                break;
                                                        case T_ROLL:
                                                                if(f_manual_type.state==0)
                                                                        {
                                                                                f_manual_type.state=1;
                                                                                f_manual_type.type=T_ROLL;
f_manual_type.parameter1=parameter1;
f_manual_type.parameter2=parameter2;
                                                                                strcpy(serial.buff_out,
```

```
"ack\r");

"\nPermission denied\n");

f_manual_type.parameter1=parameter1;
f_manual_type.parameter2=parameter2;
f_manual_type.parameter3=parameter3;
"ack\r");

"\nPermission denied\n");

f_manual_type.parameter1=parameter1;
f_manual_type.parameter2=parameter2;
f_manual_type.parameter3=parameter3;
"ack\r");

"\nPermission denied\n");

f_manual_type.type=T_PARALLEL;
f_manual_type.parameter1=parameter1;
"ack\r");

"\nPermission denied\n");

f_manual_type.type=T_GO_UNTIL;
f_manual_type.parameter1=parameter1;
f_manual_type.parameter2=parameter2;
f_manual_type.parameter3=parameter3;
"ack\r");

"\nPermission denied\n");

f_manual_type.type=T_CROSS_DOOR;
"ack\r");
```



```
                serial.send=TRUE;
        }
        else
                {
                    printf(                    ...
                };
    break;
case T_CURVE:
    if(f_manual_type.state==0)
        {
                f_manual_type.state=1;
                f_manual_type.type=T_CURVE;
                                                ...
                                                ...
                                                ...
                                                ...
                strcpy(serial.buff_out,         ...
                serial.send=TRUE;
        }
        else
                {
                    printf(                    ...
                };
    break;

case T_CIRC:
    if(f_manual_type.state==0)
        {
                f_manual_type.state=1;
                f_manual_type.type=T_CIRC;
                                                ...
                                                ...
                                                ...
                                                ...
                strcpy(serial.buff_out,         ...
                serial.send=TRUE;
        }
        else
                {
                    printf(                    ...
                };
    break;
case T_PARALLEL:
    if(f_manual_type.state==0)
        {
                f_manual_type.state=1;
                                                ...
                                                ...
                strcpy(serial.buff_out,         ...
                serial.send=TRUE;
        }
        else
                {
                    printf(                    ...
                };
    break;
case T_GO_UNTIL:
    if(f_manual_type.state==0)
        {
                f_manual_type.state=1;
                                                ...
                                                ...
                                                ...
                                                ...
                strcpy(serial.buff_out,         ...
                serial.send=TRUE;
        }
        else
                {
                    printf(                    ...
                };
    break;
case T_CROSS_DOOR:
    if(f_manual_type.state==0)
        {
                f_manual_type.state=1;
                                                ...
                strcpy(serial.buff_out,         ...
                serial.send=TRUE;
        }
        else
                {
                    printf(                    ...
```

```
"\nPermission denied\n");                                                                    715
                                                                                    };
                                                                            break;
                                                                    case T_GO_PARALLEL:
                                                                            if(f_manual_type.state==0)
                                                                                    {
                                                                                            f_manual_type.state=1;
                                                                                                                                    ...
f_manual_type.type=T_GO_PARALELL;
                                                                                                                                    ...
f_manual_type.parameter1=parameter1;
                                                                                                                                    ...
f_manual_type.parameter2=parameter2;
                                                                                                                                    ...
f_manual_type.parameter3=parameter3;                                                                                                ...
                                                                                            strcpy(serial.buff_out,                 ...
"ack\r");
                                                                                            serial.send=TRUE;
                                                                                    }
                                                                                    else
                                                                                            {
                                                                                                 printf(                            ...
"\nPermission denied\n");
                                                                                            };
                                                                            break;
                                                                    case T_GO_AT:
                                                                            if(f_manual_type.state==0)
                                                                                    {
                                                                                            f_manual_type.state=1;
                                                                                            f_manual_type.type=T_GO_AT;
                                                                                                                                    ...
f_manual_type.parameter1=parameter1;
                                                                                                                                    ...
f_manual_type.parameter2=parameter2;
                                                                                                                                    ...
f_manual_type.parameter3=parameter3;
                                                                                                                                    ...
f_manual_type.parameter4=parameter4;                                                                                                ...
                                                                                            strcpy(serial.buff_out,                 ...
"ack\r");
                                                                                            serial.send=TRUE;
                                                                                    }
                                                                                    else
                                                                                            {
                                                                                                 printf(                            ...
"\nPermission denied\n");
                                                                                            };
                                                                            break;
                                                                    case T_ACTIVE_US:
                                                                            if(f_manual_type.state==0)
                                                                                    {
                                                                                            f_manual_type.state=1;
                                                                                                                                    ...
f_manual_type.type=T_ACTIVE_US;
                                                                                                                                    ...
f_manual_type.parameter1=parameter1;
                                                                                                                                    ...
f_manual_type.parameter2=parameter2;
                                                                                                                                    ...
f_manual_type.parameter3=parameter3;
                                                                                                                                    ...
f_manual_type.parameter4=parameter4;
                                                                                            strcpy(serial.buff_out,                 ...
"ack\r");
                                                                                            serial.send=TRUE;
                                                                                    }
                                                                                    else
                                                                                            {
                                                                                                 printf(                            ...
"\nPermission denied\n");
                                                                                            };
                                                                            break;
                                                                    default:
                                                                            printf("\nInvalid task type\n");
                                                                            break;
                                                                                };
                                                            default:
                                                                    strcpy(serial.buff_out,"error\r");
                                                                    serial.send=TRUE;
                                                                    return;
                                                            };
                                                    break;
/*****************************************************************************/
                                    case MSGC_DATAQUERY:
                                            switch(type)
                                                    {
                                                            case IN_POST:
                                                                    sprintf(aux,"S %ld %ld %ld %d %ld\r"                            ...
,pos.x,pos.y,pos.theta,mov.status,get_time());
                                                                    strcpy(serial.buff_out,aux);
                                                                    serial.send=TRUE;
                                                                    break;
                                                            case IN_US:
                                                                    strcpy(serial.buff_out,"U");
                                                                    for (p=0; p<MAXSENS; p++)
                                                                            {
                                                                                    sprintf(aux," %ld",sens.values[p]);
                                                                                    strcat(serial.buff_out,aux);
                                                                            }
                                                                    sprintf(aux," %d %ld\r",mov.status,get_time());
                                                                    strcat(serial.buff_out,aux);
                                                                    serial.send=TRUE;
                                                                    break;
                                                            case IN_POSTUS:
                                                                    sprintf(serial.buff_out,"V %ld %ld %ld %d"                      ...
,pos.x, pos.y, pos.theta, mov.status);
```

```
                                        for (p=0; p<MAXSENS; p++)                                                    800
                                                {
                                                        sprintf(aux," %ld",sens.values[p]);
                                                        strcat(serial.buff_out,aux);
                                                }
                                        sprintf(aux," %ld\r",get_time());
                                        strcat(serial.buff_out,aux);
                                        serial.send=TRUE;
                                        break;
                                case IN_CLOCK:
                                        sprintf(serial.buff_out,"C %ld\r",get_time());
                                        serial.send=TRUE;
                                        break;

                                case IN_USTOUT:
                                        sprintf(serial.buff_out,"T %d\r", sens.time_out);
                                        serial.send=TRUE;
                                        break;
                                case IN_PTU_US:
                                        strcpy(serial.buff_out,"P");
                                        for (p=0; p<MAX_PTU_SENS; p++)
                                                {
                                                        sprintf(aux," %ld",pt_values[p]);
                                                        strcat(serial.buff_out,aux);
                                                }
                                        sprintf(aux," %d %ld\r",mov.status,get_time());
                                        strcat(serial.buff_out,aux);
                                        serial.send=TRUE;
                                        break;
                                case IN_PTU_FUS:
                                        strcpy(serial.buff_out,"F");
                                        for (p=0; p<MAX_PTU_SENS; p++)
                                                {
                                                        sprintf(aux," %ld",fvalues[p]);
                                                        strcat(serial.buff_out,aux);
                                                }
                                        sprintf(aux," %d %ld\r",mov.status,get_time());
                                        strcat(serial.buff_out,aux);
                                        serial.send=TRUE;
                                        break;
                                case IN_PTU_DUS:
                                        strcpy(serial.buff_out,"D");
                                        for (p=0; p<MAX_PTU_SENS; p++)
                                                {
                                                        sprintf(aux," %ld",dvalues[p]);
                                                        strcat(serial.buff_out,aux);
                                                }
                                        sprintf(aux," %d %ld\r",mov.status,get_time());
                                        strcat(serial.buff_out,aux);
                                        serial.send=TRUE;
                                        break;

                                default:
                                        strcpy(serial.buff_out,"error\r");
                                        serial.send=TRUE;
                                        return;
                        }

                        break; /* end of case MSGC_DATAQUERY: */
/*********************************************************/
                        case MSGC_UNIDIRCOM:
                            switch(type)
                                {
                                case IN_VELOCITY:
                                        if (n_param!=2)
                                                {
                                                        strcpy(serial.buff_out,"error\r");
                                                        serial.send=TRUE;
                                                        return;
                                                }
                                        mov.left=atoi(param[0]);
                                        mov.right=atoi(param[1]);
                                        emerg.rtm=TRUE;
                                        strcpy(serial.buff_out,"ack\r");
                                        serial.send=TRUE;
                                        break;
                                case IN_SERV:
                                        if ( (strcmp(param[0],"F")!=0) && (strcmp(param[0],"N")!=0) )
                                                {
                                                        strcpy(serial.buff_out,"error\r");
                                                        serial.send=TRUE;
                                                        return;
                                                }
                                        if(strcmp(param[0],"N")==0)
                                                mov.status=1;
                                        else
                                                mov.status=0;
                                        strcpy(serial.buff_out,"ack\r");
                                        serial.send=TRUE;
                                        break;
                                case IN_ODOM:
                                        sendf("ODOS C=%d,%d,%d\r"                                                    ...
,atoi(param[0]),atoi(param[1]),atoi(param[2]));
                                        strcpy(serial.buff_out,"ack\r");
                                        serial.send=TRUE;
                                        break;
                                case IN_ALBATROS:
                                        sendf("%s",param[0]);
                                        strcpy(serial.buff_out,"ack\r");
                                        serial.send=TRUE;
                                        break;
                                case IN_KILL:
                                        strcpy(serial.buff_out,"ack\r");
                                        serial.send=TRUE;
                                        s_close(serial.desc);
                                        kill=TRUE;
                                        break;
                                case IN_DOOR:
```

```
                door=ON;
                sprintf(serial.buff_out,"ack\r");
                serial.send=TRUE;
                break;
        case IN_DOOR_OFF:
                door=OFF;
                ptu_on=OFF;
                emerg.on=BUMPER+US+COM;
                sprintf(serial.buff_out,"ack\r");
                serial.send=TRUE;
                break;
        case IN_PTU_ON:
                ptu_on=ON;
                sprintf(serial.buff_out,"ack\r");
                serial.send=TRUE;
                break;
        case IN_PTU_OFF:
                ptu_on=OFF;
                sprintf(serial.buff_out,"ack\r");
                serial.send=TRUE;
                break;
        case IN_AUTO_ON:
                auto_detect=ON;
                sprintf(serial.buff_out,"ack\r");
                serial.send=TRUE;
                break;
        case IN_AUTO_OFF:
                auto_detect=OFF;
                sprintf(serial.buff_out,"ack\r");
                serial.send=TRUE;
                break;
        default:
                strcpy(serial.buff_out,"error\r");
                serial.send=TRUE;
                return;
        }
    break;
/*****************************************************/
    case MSGC_USACTION:
        switch(type)
        {
        case IN_USTABLE:
                if (strlen(param[0])<24)
                        {
                                strcpy(serial.buff_out,"error\r");
                                serial.send=TRUE;
                                return;
                        }
                token=param[0];
                for (p=0; p<24; p++)
                        {
                                sens.active_sensors[p]=*token-48;
                                token++;
                        }
                emerg.us_change=TRUE;
                strcpy(serial.buff_out,"ack\r");
                serial.send=TRUE;
                break;
        case IN_NDELAY:
                if (strlen(param[0])==0)
                        {
                                strcpy(serial.buff_out,"error\r");
                                serial.send=TRUE;
                                return;
                        }
                sens.node_delay=atoi(param[0]);
                strcpy(serial.buff_out,"ack\r");
                serial.send=TRUE;
                break;
        case IN_GNDELAY:
                sprintf(serial.buff_out,"N %d\r",sens.node_delay);
                serial.send=TRUE;
                break;
        case IN_STAT:
                if (*param[0]=='N')
                        {
                                sprintf(serial.buff_out,"S %ld\r"                    ...
,sens.stat.node_mean_time);
                        }
                        else
                                if (*param[0]=='S')
                                        {
                                                sprintf(serial.buff_out,"S %ld\r"    ...
,sens.stat.scan_time);
                                        }
                                        else
                                                strcpy(serial.buff_out,"error\r");
                        serial.send=TRUE;
                break;
        case IN_SUSTOUT:
                sens.time_out=atoi(param[0]);
                sprintf(serial.buff_out,"ack\r");
                serial.send=TRUE;
                break;
        case IN_TOGSENS:
                if (strlen(param[0])==0)
                        {
                                strcpy(serial.buff_out,"error\r");
                                serial.send=TRUE;
                                return;
                        }
                if (param[0][0]=='0')
                        emerg.auto_toggle=FALSE;
                else
                        if (param[0][0]=='1')
                                emerg.auto_toggle=TRUE;
                        else
                                {
```

```
                                                                    strcpy(serial.buff_out,"error\r");          1013
                                                                    serial.send=TRUE;
                                                                    return;
                                                                }
                                                        sprintf(serial.buff_out,"ack\r");
                                                        serial.send=TRUE;
                                                        break;
                                                default:
                                                        strcpy(serial.buff_out,"error\r");
                                                        serial.send=TRUE;
                                                        return;
                                        }
                                        break;
/*************************************************/
                default:
                        strcpy(serial.buff_out,"error\r");
                        serial.send=TRUE;
                        return;
        }
}
/*****************************************************************************************************/
int init_serial_spy(void)
{

        init_port();

        init_mission_array();

        if ( (serial.desc=s_open(PORT,O_RDWR,P2NOECHO)) <= 0)
                {
                        printf("\nERROR opening serial port!\n\r");
                        return(ERROR);
                }

        emerg.time_stamp=get_time();
        return(OK);
}
/*****************************************************************************************************/

void close_serial_spy(void)
{

        s_close(serial.desc);
        kill_port();
}
/*****************************************************************************************************/
void serial_spy(void)
{
        static int first_time=TRUE;
        static char *token;
        static char ch;
        int size;

        if (first_time)
                {
                        token=serial.buff_in;
                        first_time=FALSE;
                }

        size=s_read(serial.desc,&ch,1);
        if (size==0)
                return;
        else
                if ((ch!=TERMINATOR)&&(ch!=TERMINATOR2))
                        {
                                printf("%c",ch);
                                *token=ch;
                                token++;
                                return;
                        }

        *token=TERMINATOR;
        token++;
        *token='\0';
        emerg.time_stamp=get_time();

        decode();
        token=serial.buff_in;

        if(kill == FALSE)
                printf("\n\r->");

        if (serial.send)
                {
                        size=strlen(serial.buff_out);
                        s_write(serial.desc,serial.buff_out,size);
                        serial.send=FALSE;
                }
}
/*****************************************************************************************************/

void init_mission_array(void)
{
        int i,j;

        free_cell=0;
        tail=0;

        for(i=0;i<=MAX_TASKS;i++)
                mission_manager[i]=0;

        position_array=0;
        for(i=0;i<=MAX_TASKS;i++)
                {
                        for(j=0;j<=MAX_PARAMETERS;j++)
                                mission_array[i].parameter[j]=0;
                        mission_array[i].next=0;
                        mission_array[i].state=0;
```

```
                };
};/*end of init_mission_array */
/*****************************************************************************************/

void print_task(int i)
{
        int j,n;
        int aux1=0;

        n=mission_manager[i];

        switch(mission_array[n].parameter[0])
                {
                        case T_GO:
                           printf(" %d  go(",n);
                           aux1=1;
                           break;
                        case T_ROLL:
                           printf(" %d  roll(",n);
                           aux1=1;
                           break;
                        case T_CURVE:
                           printf(" %d  curve(",n);
                           aux1=1;
                           break;
                        case T_CIRC:
                           printf(" %d  circ(",n);
                           aux1=1;
                           break;
                        case T_PARALLEL:
                           printf(" %d  parallel(",n);
                           aux1=1;
                           break;
                        case T_GO_UNTIL:
                           printf(" %d  go_until(",n);
                           aux1=1;
                           break;
                        case T_CROSS_DOOR:
                           printf(" %d  cross_door(",n);
                           aux1=1;
                           break;
                        case T_GO_PARALELL:
                           printf(" %d  go_parallel(",n);
                           aux1=1;
                           break;
                        case T_GO_AT:
                           printf(" %d  go_at(",n);
                           aux1=1;
                           break;
                        case T_ACTIVE_US:
                           printf(" %d  active_us(",n);
                           aux1=1;
                           break;
                };
                   for(j=1;j<=4;j++)
                        if(mission_array[n].parameter[j]!= 0)
                                {
                                   if(mission_array[n].parameter[0] == T_ACTIVE_US)
                                        {
                                                printf("%ld",mission_array[n].parameter[j]);
                                        }
                                        else
                                                switch(mission_array[n].parameter[j])
                                                        {
                                                                case FRONT:
                                                                        printf("FRONT");
                                                                        break;
                                                                case RIGHT:
                                                                        printf("RIGHT");
                                                                        break;
                                                                case BACK:
                                                                        printf("BACK");
                                                                        break;
                                                                case LEFT:
                                                                        printf("LEFT");
                                                                        break;
                                                                default:
                                                                        printf("%ld",mission_array[n].parameter[j]);
                                                                        break;
                                                        };
                                   if(j<4)
                                        if(mission_array[n].parameter[j+1]!= 0)
                                                printf(",");
                                };
        if(aux1>0)
                printf(");\n");
/*      printf("    next -> %d      previous->%d",mission_array[i].next,detect_previous(i));*/

}; /* end print_task */
/*****************************************************************************************/
int detect_first_empty(void)
{
        int i=-1;
        int end=0;

        end=1;
        while(end==1)
                {
                        i++;
                        if(mission_array[i].state != 1)
                                {
                                        end=0;
                                };
                };
        return(i);
};
/*****************************************************************************************/
```

```
int detect_previous(int i)
{
        int j=0;
        int end=1;

        if(i!= 0)
                while(end==1)
                        {

                                if(mission_array[j].next == i)
                                end=0;
                                j++;
                        }
                else
                        j=1;

        return(j-1);
};
/*****************************************************************************************/

void actualize_mission_manager(void)
{
        int i;
        int end=1;

        for(i=0;i<= MAX_TASKS;i++)
                mission_manager[i]=0;

        if(mission_array[position_array].state == 1)
                mission_manager[0]=position_array;

        i=1;
        while(end == 1)
                {
                        if((mission_array[mission_manager[i-1]].next == 0) || (i>=MAX_TASKS))
                                end=0;
                        mission_manager[i]=mission_array[mission_manager[i-1]].next;
                        i++;
                };
};
/*****************************************************************************************/
```

```
#define TIME_BEFORE_MOVE 400
#define MAX_DIST 1000
#define ROBOT_LENGTH 1025
#define ROBOT_WIDTH 700

#define PAN_X_POSITION 725   /* Distance to the front */

#define PAN_Y_POSITION 350   /* Distance to the left side */
#define BOX_WIDTH 100
#define SAFETY_DIST 40     /* era 35 */


/* inicilization of process for crossing doors */
void init_doors(void);

/* Process that controls doors crossing */
void doors(void);
```

```
#include <libcstd.h>
#include <libport.h>
#include <libcalbl.h>
#include <libcpro.h>
#include <libport.h>

#include "doors.h"
#include "serial.h"
#include "generic.h"
#include "headers.h"
#include "ptu.h"
#include "executor.h"
#include "emerg.h"
#include "us.h"

extern long pt_values[MAX_PTU_SENS];
extern long dvalues[MAX_PTU_SENS];
extern MOVES mov;
extern EMERGENCY emerg;
extern US_DATA sens;
extern ptu_on;

int door;
int auto_detect;
int success;
long min_read[MAX_PTU_SENS];



  static long int time=0;
  static int count=0;

  static int inside_door=FALSE;
  static int speed_in_door0=2;
  static int speed_in_door1=-1;
  static int speed_in_door2=0;

/*****************************************************************************/
void init_doors(void)
{
  int x1,x2,x3,x4,i;

  door=OFF;
  success=FALSE;

  x1=PAN_X_POSITION-BOX_WIDTH/2+SAFETY_DIST;
  x2=ROBOT_LENGTH-PAN_X_POSITION-BOX_WIDTH/2+SAFETY_DIST;
  x3=PAN_Y_POSITION-BOX_WIDTH/2+SAFETY_DIST;
  x4=ROBOT_WIDTH-PAN_Y_POSITION-BOX_WIDTH/2+SAFETY_DIST;

  min_read[0]=x1;
  min_read[36]=x2;

  for(i=1;i<=5;i++)
        min_read[i]=x1/cos((5*3.14/180)*i);

  for(i=6;i<=18;i++)
        min_read[i]=x3/sin((5*3.14/180)*i);

  for(i=19;i<=27;i++)
        min_read[i]=x3/sin((5*3.14/180)*(36-i));

  for(i=28;i<=35;i++)
        min_read[i]=x2/cos((5*3.14/180)*(36-i));

  for(i=37;i<=45;i++)
        min_read[i]=x2/cos((5*3.14/180)*(i-36));

  for(i=46;i<=54;i++)
        min_read[i]=x4/sin((5*3.14/180)*(i-36));

  for(i=55;i<=66;i++)
        min_read[i]=x4/sin((5*3.14/180)*(72-i));

  for(i=67;i<=71;i++)
        min_read[i]=x1/cos((5*3.14/180)*(72-i));
}
/*****************************************************************************/
void end_doors(void)
{

  int i;

  door=OFF;
  ptu_on=OFF;
  emerg.on=BUMPER+US+COM;
  speed_in_door0=2;
  speed_in_door1=-1;
  speed_in_door2=0;
  inside_door=FALSE;
  mov.left=0;
  mov.right=0;
  mov.flag=TRUE;
  for (i=0; i<24; i++)                    /* Make all sensors inactive */
  sens.active_sensors[i]=1;
  sens.change_flag=TRUE;
}

/*****************************************************************************/
int detect_door(void)
  {
  int i;

  if (auto_detect==OFF)
        return(-1);

  if (sens.values[0]>MAX_DIST)
  {
```

```
   for(i=1;i<=5;i++)
         if(sens.values[i]<600)
           {
           for(i=19;i<=23;i++)
                if(sens.values[i]<600)
           return(0);
       break;
           }
  }
  return(-1);
  }

/*****************************************************************************/
void doors(void)
  {

        int turn_left,turn_right;
        int i,j;

        if ( (detect_door()>=0)&&(door==OFF) )
                {
                        time=get_time();
                        count=0;
                        success=FALSE;
                        ptu_on=ON;
                        door=ON;

                        for (i=0; i<24; i++)                /* Make all sensors inactive */
                                sens.active_sensors[i]=0;
                        emerg.on=BUMPER;
                        sens.change_flag=TRUE;
                }

        if (get_time()-time<TIME_BEFORE_MOVE)
                {
                        mov.left=2;
                        mov.right=2;
                        return;
                }
        if (door==ON)
                {
                        if(inside_door==FALSE)
                                {
                                        for(i=5;i<=17;i++)
                                                if(pt_values[i]<min_read[i]+200)
                                                        {
                                                                i=17;
                                                                for(j=54;j<=67;j++)
                                                                if(pt_values[j]<min_read[i]+200)
                                                                        {
                                                                                speed_in_door0=4;
                                                                                speed_in_door1=2;
                                                                                speed_in_door2=4;
                                                                                inside_door=TRUE;
                                                                                j=65;
                                                                        }
                                                        }
                                }
                        turn_left=FALSE;
                        turn_right=FALSE;
                        for(i=1;i<=17;i++)
                        if(pt_values[i]<min_read[i])
                                {
                                        turn_right=TRUE;
                                }
                        for(i=54;i<=70;i++)
                                if(pt_values[i]<min_read[i])
                                        {
                                                turn_left=TRUE;
                                        }
                        if((turn_left==FALSE)&&(turn_right==FALSE))
                                {
                                        count=0;
                                        mov.left=speed_in_door0;
                                        mov.right=speed_in_door0;
                                        mov.flag=TRUE;
                                }
                        else if((turn_left==TRUE)&&(turn_right==FALSE))
                                        {
                                                count=0;
                                                mov.left=speed_in_door1;
                                                mov.right=speed_in_door2;
                                                mov.flag=TRUE;
                                        }
                        else if((turn_left==FALSE)&&(turn_right==TRUE))
                                        {
                                                count=0;
                                                mov.left=speed_in_door2;;
                                                mov.right=speed_in_door1;
                                                mov.flag=TRUE;
                                        };
        /*              else
                                {
                                        count++;
                                        mov.left=0;
                                        mov.right=0;
                                        mov.flag=TRUE;
                                        if (count>((TIME_BEFORE_MOVE)/10))
                                                {
                                                        end_doors();
                                                        printf("Can't cross the door");
                                                }
                                }*/

                        for(i=1;i<=17;i++)
                                if(pt_values[i]<(MAX_DIST))
                                        {
                                                for(i=54;i<=71;i++)
```

```
                                        if(pt_values[i]<(MAX_DIST))
                                                return;
                                        end_doors();
                                        success=TRUE;
                                        return;
                                }
                        end_doors();
                        success=TRUE;

                }
}
/*****************************************************************************/
```

```
#define ERROR       -1
#define OK           1

#define SPEED 2000
#define PAN_TIME_OUT 15
#define PAN_TIME_OUT_DIST 2491

#define MAX_PTU_SENS 72


/* Move the pan axis to the absolute position pos at the speed desired */
void move_ptu(int pos,int speed);

/* Process that coordinates PTU movements and read angle value */
void ptu(void);

void send2ptu(char *);

int init_ptu();

void close_ptu();
```

```
#include <libcstd.h>
#include <libport.h>
#include <libcalbl.h>
#include <libcpro.h>
#include <libport.h>

#include "ptu.h"
#include "serial.h"
#include "generic.h"
#include "headers.h"
#include "us.h"

#define PORT "SLD"
#define MAX 2300

static SERIAL ptu_serial;


long pt_values[MAX_PTU_SENS];
long fvalues[MAX_PTU_SENS];
long dvalues[MAX_PTU_SENS];

int ptu_on;

static int angle;
int ccc;
int local_angle;

int div10(int value)
{
        char aux[6];

        if (value == 0)
                return(0);

        sprintf(aux, "%d", value);
        if (strlen(aux) < 2)
                return(0);

        aux[strlen(aux)-1]='\0';
        return(atoi(aux));
};
/*****************************************************************************/
void filter(void)
{
        int i;

        for (i=0; i<70; i++)
                {
                        if ( (pt_values[i]>MAX) && (pt_values[i+1]<MAX-1000) && (pt_values[i+2]>MAX) )
                                fvalues[i+1]=(pt_values[i]+pt_values[i+2])/2;
                                        else
                                                fvalues[i+1]=pt_values[i+1];
                };
        if ( (pt_values[70]>MAX) && (pt_values[71]<MAX-1000) && (pt_values[0]>MAX) )
                fvalues[71]=(pt_values[70]+pt_values[0])/2;
                        else
                                fvalues[71]=pt_values[71];

        if ( (pt_values[71]>MAX) && (pt_values[0]<MAX-1000) && (pt_values[1]>MAX) )
                fvalues[0]=(pt_values[71]+pt_values[1])/2;
                        else
                                fvalues[0]=pt_values[0];
};
/*****************************************************************************/
void diff(void)
{
        int i;

        filter();

        for (i=0; i<71; i++)
                dvalues[i]=fvalues[i+1]-fvalues[i];
        dvalues[71]=fvalues[0]-fvalues[71];
};
/*****************************************************************************/
void read_pt(void)
{

        static long temp[6];
        long ret;
        char aux[5];
        int pos;
        static int first_time=TRUE;

        local_angle=angle;
        sprintf(aux, "%d", local_angle);
        switch (aux[strlen(aux)-1])
                {
/*                      case'4':*/

                        case '9':
                                if(local_angle>0)
                                        local_angle++;
                                        else
                                                local_angle--;
                                break;

/*                      case'6':*/

                        case '1':
                                if(local_angle>0)
                                        local_angle--;
                                        else
                                                local_angle++;
                                break;
                };
        sprintf(aux, "%d", local_angle);
```

```
        if ( (aux[strlen(aux)-1] == '5') || (aux[strlen(aux)-1] == '0') )                    109
                {
                        pos=2*(local_angle+90);
                        pos=pos/10;
                        if (first_time == TRUE)
                                {
                                        sendf("READ C=7,110000 T=%d V=%d O=%lx\r", PAN_TIME_OUT, SOUND_SPEED, temp);
                                        first_time=FALSE;
                                };
                        do
                                {
                                        ret=sendf("READ C=7,110000 T=%d V=%d O=%lx\r",PAN_TIME_OUT, SOUND_SPEED ,temp);
                                } while ( ret!=0);

/* Values greather than time out are equal to time_out */
                        if(temp[0]>PAN_TIME_OUT_DIST)
                                temp[0]=PAN_TIME_OUT_DIST;
                        if(temp[1]>PAN_TIME_OUT_DIST)
                                temp[1]=PAN_TIME_OUT_DIST;
/* Os valores sao corrigidos para uma velocidade do som de 335 */
                        if(temp[1]>=0)
                                pt_values[pos]=(temp[1]-46)/0.96;
                        if(temp[0]>=0)
                                pt_values[pos+36]=(temp[0]-46)/0.96;
                };
}; /* end of read_pt */
/*****************************************************************************/

void send2ptu(char *msg)
{
        char aux[50];
        char ch;

        strcpy(aux, msg);
        strcat(aux, "\r");
        s_write(ptu_serial.desc,aux,strlen(aux));
        do
                s_read(ptu_serial.desc,&ch,1);
        while(ch!=13);
};
/*****************************************************************************/

int init_ptu(void)
{
        int n;

        if ( (ptu_serial.desc=s_open(PORT,O_RDWR) ) <= 0)
                {
                        /*  printf("\nERROR opening PTU serial port!\n\r");*/
                        return(ERROR);
                };

        /* printf("\nPTU started.\n");*/

        send2ptu("ED");
        send2ptu("FT");
        send2ptu("I");
        send2ptu("PA6000");
        move_ptu(-1755,SPEED);
        for (n=0; n<MAX_PTU_SENS; n++)
                {
                        pt_values[n]=0;
                        fvalues[n]=0;
                        dvalues[n]=0;
                };
        ptu_on=FALSE;
        return(OK);
}; /* end of init_ptu() */
/*****************************************************************************/

void close_ptu(void)
{
        /* printf("\n--> %d", ccc);*/
        s_close(ptu_serial.desc);
};
/*****************************************************************************/

void move_ptu(int pos, int speed)
{
        static int last_speed=0;
/* Set the speed of the pan & tilt unit */
        if (speed != last_speed)
                {
                        sprintf(ptu_serial.buff_out,"ps%d\r",speed);
                        send2ptu(ptu_serial.buff_out);
                        last_speed=speed;
                };
/* Send the movement to the monitor of the pan & tilt unit */
        sprintf(ptu_serial.buff_out,"pp%d\r",pos);
        send2ptu(ptu_serial.buff_out);
};
/*****************************************************************************/

void ptu(void)
{

        int size;
        static char *token;
        static int state=1;
        static int first_time=TRUE;
        static int position=0;
        int n;
        char ch;

        if (ptu_on)
                {
                        diff();
                        if (first_time)
```

```
                {
                        s_write(ptu_serial.desc,"PP\r",3);
                        ccc=0;
                        first_time=FALSE;
                        return;
                };
        if ( (state==0) && (position==1655) )
                {
                        state=1;
                        move_ptu(-1755,SPEED);
                        do
                                s_read(ptu_serial.desc,&ch,1);
                        while(ch!=13);
                };
        if ( (state==1) && (position==-1755) )
                {
                        move_ptu(1655,SPEED);
                        state=0;
                        do
                                s_read(ptu_serial.desc,&ch,1);
                        while(ch!=13);
                };
        size=s_read(ptu_serial.desc,ptu_serial.buff_in,20);
        if (size==0)
                return;
        token=ptu_serial.buff_in;
        while (*token!=13)
                token++;
        *token='\0';
        token=ptu_serial.buff_in+2;
        position=atoi(token);
        angle=position*185/3600;
        read_pt();
        s_write(ptu_serial.desc,"PP\r",3);
}
else
        {
                first_time=FALSE;
                for (n=0; n<MAX_PTU_SENS; n++)
                        pt_values[n]=0;
        };
}; /* end of ptu */
/*****************************************************************************/
```

```
/* #define US_LIMIT    1100    Flight time in microseconds */

/* #define US_LIMIT    220    Distance in mm */
/* Security distance added to normal stop distance */

#define COMM_LIMIT     180000     /* 30 minute(s) */

#define NOT_PRESSED    1          /* Bumper state */
#define PRESSED        0

#define DBW   600     /* Distance between wheels */

#define LIMIT12    2476
#define LIMIT23    1200
#define LIMIT34    1011
#define LIMIT45    864
#define LIMIT56    710

#define NONE 0
#define BUMPER 1
#define US 2
#define COM 4

typedef struct {
  int flag;        /* Indicates an emergency execption */
  int rtm;         /* Request To Move */
  int us_change;  /* Request to change AST */
  long time_stamp;
  int auto_toggle;
  int on;
} EMERGENCY;

/* Subtract two sensor numbers */
int sub24(int, int);


/* Add two sensor numbers */
int add24(int, int);


/* Returns bumper state */
int get_bumper_state(void);

/* stop in case of emergency */
void emergency_stop(void);


/* Returns main sensor given the actual orientation of motion */
int get_main_sensor(void);


/* Make all initializations in data structs used by emergecy process */
void init_emergency(void);


/* Emergency process */
void emergency(void);
```

```
#include <libcrobu.h>
#include <libcalbl.h>

#include "emerg.h"
#include "generic.h"
#include "executor.h"
#include "serial.h"
#include "us.h"
#include "headers.h"


EMERGENCY emerg;

extern MOVES mov;
extern POSTURE pos;


int left_vel=0, right_vel=0;

int get_us_limit() {
  int max;

  if (abs(mov.left) > abs(mov.right))
    max=abs(mov.left);
  else
    max=abs(mov.right);

  if (max<22)
     return(220);
  return(max*10);
}


int sub24(int a, int b) {
  int r;

  r=a-b;
  if (r<1)
    r=24+r;
  return r;
}


int add24(int a, int b) {
  int r;

  r=a+b;
  if (r>24)
    r=r-24;
  return r;
}


void set_table(int *a, int *b) {
  int n;
  int *source, *dest;

  dest=a;
  source=b;

  for (n=1; n<=24; n++) {
    *dest=*source;
    dest++;
    source++;
  }
}


void table_or(int *a, int *b) {
  int n;
  int *source, *dest;

  dest=a;
  source=b;

  for (n=1; n<=24; n++) {
    if ( (*dest==ACTIVE) || (*source==ACTIVE) )
      *dest=ACTIVE;
    else
      *dest=NOT_ACTIVE;
    dest++;
    source++;
  }
}


int get_bumper_state() {
  static int first_time=TRUE;
  static long *pl;

  if (first_time==TRUE) {
    pl=get_log_input(1);
    first_time=FALSE;
  }
  return(*pl & 1);
}


int get_main_sensor(void) {
  int left, right;
  double r;

  /* Nao tem velocidades armazenadas */
  if ( (left_vel==0) && (right_vel==0) ) {
    left=mov.left;
    right=mov.right;
  }
```

```
  else {
    left=left_vel;
    right=right_vel;
  }

  if (left==right) {
    if (left==0)
      return(0);                    /* No motion */
    else if (left<0)
      return(13);                   /* Backward motion */
    else
      return(1);                    /* Forward motion */
  }

  r = (left+right)*1.0/(left-right);
  r = r*DBW/2;

  if (fabs(r) > LIMIT12) {          /* Straight motion */
    if ( (left>0) && (right>0) ) {
      return(1);
    }
    else {
      return(13);
    }
  }
  else if (fabs(r) > LIMIT23) {     /* Very soft curve */
    if ( (left>0) && (right>0) ) {
      if (r > 0)
        return(2);
      else
        return(24);
    }
    else {
      if (r > 0)
        return(12);
      else
        return(14);
    }
  }
  else if (fabs(r) > LIMIT34) {     /* Soft curve */
    if ( (left>0) && (right>0) ) {
      if (r > 0)
        return(3);
      else
        return(23);
    }
    else {
      if (r > 0)
        return(11);
      else
        return(15);
    }
  }
  else if (fabs(r) > LIMIT45) {     /* Medium curve */
    if ( (left>0) && (right>0) ) {
      if (r > 0)
        return(4);
      else
        return(22);
    }
    else {
      if (r > 0)
        return(10);
      else
        return(16);
    }
  }
  else if (fabs(r) > LIMIT56) {     /* Hard curve */
    if ( (left>0) && (right>0) ) {
      if (r > 0)
        return(5);
      else
        return(21);
    }
    else {
      if (r > 0)
        return(9);
      else
        return(17);
    }
  }
  else if (r==0) {                  /* Rotation */
    if (left>0)
      return(7);
    else
      return(19);
  }
  else if (left>right) {            /* Very hard curve */
    if (fabs(left) > fabs(right))
      return(6);
    else
      return(18);
  }
  else {
    if (fabs(left) > fabs(right))
      return(8);
    else
      return(20);
  }
}

void emergency_stop(void)
  {
/*      int aux_w=0;

        if((mov.left>40) || (mov.right>40))
                aux_w=-10;
        sendf("MOVE V AC=%d,%d\r",mov.left/2,mov.right/2);
```

```
        delay(12-aux_w);
        sendf("MOVE V AC=%d,%d\r",mov.left/2,mov.right/3);
        delay(12-aux_w);
        sendf("MOVE V AC=%d,%d\r",mov.left/2,mov.right/10);
        delay(12-aux_w);
        sendf("MOVE V AC=%d,%d\r",0,0);
        delay(10);                    */
  send("ODOM OF\r");
  send("SERV OF\r");
  send("ODOM ON\r");
  sendf("ODOS C=%ld,%ld,%ld\r", pos.x, pos.y, pos.theta);
  }


void init_emergency(void) {
  send("BUMP OF\r");
  emerg.flag=NONE;
  emerg.rtm=FALSE;
  emerg.us_change=FALSE;
  emerg.auto_toggle=FALSE;
  emerg.time_stamp=get_time();
  emerg.on=BUMPER+US+COM;
}


void emergency(void) {
  static int wds[5];
  static int emergency_active_sensors[24];
  static int last_auto_toggle=FALSE;
  static int last_m_sensor=0;
  int main_sensor, n, i;

  extern US_DATA sens;

  main_sensor=get_main_sensor();

/*********************************************************
**********************************************************
Testa e actualiza os sensores US se esse modulo esta activo
**********************************************************
**********************************************************/
if ( (emerg.on&US)==US )
  {
/************************************************
Actualiza a tabela de sensores de emergencia activos
************************************************/
  if (main_sensor!=last_m_sensor) {

    if (main_sensor==7) {
      wds[0]=7;
      wds[1]=5;
      wds[2]=6;
      wds[3]=17;
      wds[4]=18;
    }
    else if (main_sensor==19) {
      wds[0]=19;
      wds[1]=20;
      wds[2]=21;
      wds[3]=8;
      wds[4]=9;
    }
    else if (main_sensor==6) {
      wds[0]=6;
      wds[1]=5;
      wds[2]=7;
      wds[3]=18;
      wds[4]=24;
    }
    else if (main_sensor==8) {
      wds[0]=8;
      wds[1]=7;
      wds[2]=9;
      wds[3]=14;
      wds[4]=20;
    }
    else if (main_sensor==20) {
      wds[0]=20;
      wds[1]=21;
      wds[2]=19;
      wds[3]=2;
      wds[4]=8;
    }
    else if (main_sensor==18) {
      wds[0]=18;
      wds[1]=17;
      wds[2]=19;
      wds[3]=12;
      wds[4]=6;
    }
    else if (main_sensor==0) {
      wds[0]=0;
      wds[1]=0;
      wds[2]=0;
      wds[3]=0;
      wds[4]=0;
    }
    else {
      wds[0]=main_sensor;
      wds[1]=sub24(main_sensor,1);
      wds[2]=add24(main_sensor,1);
      wds[3]=sub24(main_sensor,2);
      wds[4]=add24(main_sensor,2);
    }

    for (n=1; n<=24; n++) {            /* Constructs emergency_active_sensors */
      i=0;
      while ( (n!=wds[i]) && (i<5) )
```

```
        i++;
      if (i==5)
        emergency_active_sensors[n-1]=NOT_ACTIVE;
      else
        emergency_active_sensors[n-1]=ACTIVE;
    }

    if (emerg.auto_toggle==TRUE)
      set_table(sens.active_sensors,emergency_active_sensors);
    else
      table_or(sens.active_sensors,emergency_active_sensors);

    sens.change_flag=TRUE;
    last_m_sensor=main_sensor;
    last_auto_toggle=emerg.auto_toggle;
  }

  if ( (last_auto_toggle!=emerg.auto_toggle) || (emerg.us_change==TRUE) ) {
    if (emerg.auto_toggle==TRUE)
      set_table(sens.active_sensors,emergency_active_sensors);
    else
      table_or(sens.active_sensors,emergency_active_sensors);

    emerg.us_change=FALSE;
    sens.change_flag=TRUE;
    last_auto_toggle=emerg.auto_toggle;
  }

/*************************************/
/* Testa as emergencias nos sensores US */
/*************************************/
  if (main_sensor!=0) {
    i=0;
    while ( (sens.values[wds[i]-1]>0) && (i<5) )
      i++;
    if (i<5) {
      if ( (left_vel==0) && (right_vel==0) ) {
        left_vel=mov.left;
        right_vel=mov.right;
      }
          if (mov.status!=0) {
        mov.left=0;
        mov.right=0;
        mov.flag=TRUE;
      }
      printf("\nWARNING!  Not enough US information (missing sensor no. %d).",wds[i]);
      return;
    }
    else if ( (left_vel!=0) || (right_vel!=0) ) {
      mov.left=left_vel;
      mov.right=right_vel;
      left_vel=0;
      right_vel=0;
    }
    i=0;
    while ( (sens.values[wds[i]-1] > get_us_limit()) && (i<5) )
      i++;
    if (i<5) {
      emerg.flag=emerg.flag|US;      /* Sinaliza uma emergencia devida aos US */
          emergency_stop();
          left_vel=0;
          right_vel=0;
          emerg.rtm=FALSE;
      printf("\nWARNING!  Object too close (see sensor no. %d = %ld).",wds[i],sens.values[wds[i]-1]);
      /* printf("\nAge = %ld - Actual time = %ld",sens.node_times[get_sensor_node(wds[i])-1], get_time()); */
      return;
    }
    emerg.flag=NONE;
    if (emerg.rtm==TRUE) {
      mov.flag=TRUE;
      emerg.rtm=FALSE;
    }
  }

  }

/*************************************/
/* Testa as emergencias nos bumpers */
/*************************************/
if ( (emerg.on&BUMPER)==BUMPER )
{
  if (get_bumper_state()==PRESSED) {
    emerg.flag=emerg.flag|BUMPER;     /* Sinaliza uma emergencia devida ao bumper */
        emergency_stop();
    printf("\nWARNING! See bumper!");
    return;
  }
    else
      emerg.flag=NONE;
}

/*************************************************************************/
/* Se nao houve emergencia e ha um pedido de movimento, permite o movimento */
  if (emerg.rtm==TRUE) {    /* Request to stop */
     mov.flag=TRUE;
         emerg.rtm=FALSE;
  }

/*************************************/
/* Testa as emergencias nas comunicacoes */
/*************************************/
if ( (emerg.on&COM)==COM )
{
  if (get_time()-emerg.time_stamp > COMM_LIMIT) {
    emerg.flag=emerg.flag|COM;       /* Sinaliza uma emergencia devida as comunicacoes */
        emergency_stop();
    printf("\nWARNING!  I am alive. What about you?");
        return;
```

```
  }
}

  emerg.flag=NONE;
}
```

```
#define MAXSENS          24
#define DEFAULT_N_DELAY   3
#define DEFAULT_TIME_OUT 33
#define NOT_ACTIVE        0
#define ACTIVE            1
#define SOUND_SPEED       325


typedef struct {
  int change_flag;
  int active_sensors[24];
  int node_delay;
  int time_out;
  long values[24];
  long node_times[6];
  struct {
        long node_mean_time;
    long scan_time;
  } stat;
} US_DATA;


/* Active Sensors Table - AST */

typedef struct {
  int node[6];      /* Contains the number of active sensores in a node */
  char sens[6][7];
} ACTIVE_TABLE;


/* Gets sensor number (1..24) given Node_number and Number_in_node */
int get_sensor_number(int Node_number, int Number_in_node);


/* Performs all measures for all active sensors in the given Node_number */
void read_node(int Node_number);

void init_read_sensors(void);

/* Performs all reading for all active sensors on AST */
void read_sensors(void);


int get_sensor_node(int);
```

```
#include <libcrobu.h>
#include <libcalbl.h>

#include "us.h"
#include "generic.h"
#include "headers.h"


US_DATA sens;


static us_conf_desc *us_conf;
static ACTIVE_TABLE table;


int get_sensor_node(int number) {
  return((us_conf+number-1)->sensor/10);
}


int get_sensor_number(int node,int number)
{
  int j=0;

  while ( (us_conf+j)->sensor != node*10+number )
    j++;
  return(j+1);
}


void read_node(int node)
{
  static long temp[6][6];
  static int last_node=0;
  static int first_time=TRUE;
  long ret;
  int i;

  if (last_node!=node) {
    last_node=node;
    delay_ticks(sens.node_delay);
  }

  if (first_time==TRUE) {
    sendf("READ C=%d,%s O=%lx\r",node,table.sens[node-1],temp[node-1]);
    first_time=FALSE;
  }


  send("READ M=S R=D T=33 V=300\r");   /* Single-shot mode, timeout of 33 ms and return distance */

  do {
    ret=sendf("READ C=%d,%s O=%lx\r",node,table.sens[node-1],temp[node-1]);
        sens.node_times[node-1]=get_time();
  } while (ret != 0);

  if (sens.values[get_sensor_number(node,1)-1] == 0)    /* If this sensor was off, make an other read */
    do {
      ret=sendf("READ C=%d,%s O=%lx\r",node,table.sens[node-1],temp[node-1]);
          sens.node_times[node-1]=get_time();
    } while (ret != 0);


  for (i=0; i<4; i++) {
    sens.values[get_sensor_number(node,i+1)-1]=temp[node-1][i];
  }
}


void init_read_sensors(void) {
  int i, j;
  sens.stat.node_mean_time=0;


  /* Initialize AST (defined in US.H) */
  for (i=0;i<6;i++) {
    table.node[i]=0;
    for (j=0;j<6;j++)
      table.sens[i][j]='0';
    table.sens[i][6]='\0';
  }

  for (i=0; i<24; i++) {                /* Make all sensors active */
    sens.active_sensors[i]=1;
    sens.values[i]=0;
  }
  sens.change_flag=TRUE;                /* TRUE if the 'active_sensors' array was changed */

  sens.node_delay=DEFAULT_N_DELAY;      /* Default node firing delay */
  sens.time_out=DEFAULT_TIME_OUT;       /* Default time-out */

  send("SSET S=1 N=13\r");              /* Because sensors 2 and 24 positions */
  send("SSET S=23 N=52\r");             /*  was changed                       */

  send("READ M=S R=D T=33 V=300\r");    /* Single-shot mode, timeout of 33 ms and return distance */
  printf("\nSingle-shot Mode Start.");
  us_conf = get_us_conf(1);
}


void read_sensors()
{
  int node,number,i;
  long start, end;
  long scan_start, scan_end;
  int scan, ret;
  static total_read_time=0;
```

```
  static numb_read=0;
  static int last_time_out=DEFAULT_TIME_OUT;

  /* Fill in AST, if necessary */
  if (sens.change_flag==TRUE) {
    for (i=0;i<24;i++) {
      node=(us_conf+i)->sensor/10;
      number=(us_conf+i)->sensor-10*node;
      if (sens.active_sensors[i]!=table.sens[node-1][number-1]-48){
        if (sens.active_sensors[i]==ACTIVE) {
          table.node[node-1]++;
          table.sens[node-1][number-1]='1';
        }
        else {
          table.node[node-1]--;
          table.sens[node-1][number-1]='0';
                sens.values[i]=0;
        }
      }
    }
    sens.change_flag=FALSE;
  }

  if (sens.time_out!=last_time_out) {
        do {
      ret=sendf("READ M=S R=D T=%d V=300\r", sens.time_out);
        } while (ret!=0);
        last_time_out=sens.time_out;
  }

  /* Read nodes, if necessary */

  /* Readings without statistical data */
  /*
 for (i=1;i<=6;i++)
  if (table.node[i-1] > 0)
   read_node(i);
 */


  /* Readings with statistical data */
  scan=0;
  scan_start=get_time();
  for (i=1;i<=6;i++) {
    if (table.node[i-1] > 0) {
          scan++;
      start=get_time();
      read_node(i);
      end=get_time();
      numb_read++;
      total_read_time=total_read_time+end-start;
      sens.stat.node_mean_time=total_read_time/numb_read;
    }
  }
  scan_end=get_time();
  if (scan==6)
        sens.stat.scan_time=scan_end-scan_start;
}
```

```
typedef struct {
  long int x;
  long int y;
  long int theta;
} POSTURE;


typedef struct {
  int status;
  int left;
  int right;
  int flag;
} MOVES;

void init_execute(void);

/* Execute all actions related with motion */
void execute(void);

/* Execute serv of before leaving program*/
void close_execute(void);
```

```
#include <libcrobu.h>
#include <libcalbl.h>

#include "emerg.h"
#include "executor.h"
#include "generic.h"
#include "headers.h"


MOVES mov;
POSTURE pos;

extern EMERGENCY emerg;


void init_execute(void) {
  mov.flag=FALSE;
  mov.status=0;
  mov.left=0;
  mov.right=0;

  pos.x=0;
  pos.y=0;
  pos.theta=0;
  send("ODOM ON\r");

  printf("\nExecutor started.");
}


void close_execute(void) {
  send("SERV OF\r");
}


void execute(void) {
  static int serv_stat=0;
  static long int temp[6];

  sendf("ODOM O=%lx\r",temp);
  pos.x=temp[0];
  pos.y=temp[1];
  pos.theta=temp[2];

/*reset in case of emergency*/
  if (emerg.flag>0) {
    mov.left=0;
    mov.right=0;
        mov.status=0;
        serv_stat=0;
    mov.flag=FALSE;
        return;
  }

/*Actualization of status if necessary*/
  if (mov.status!=serv_stat) {
    if (mov.status==1)
      send("SERV ON T=BV\r");
    else {
          send("ODOM OF\r");
      send("SERV OF\r");
      mov.left=0;
      mov.right=0;
          send("ODOM ON\r");
/*        sendf("ODOS C=%ld,%ld,%ld\r", pos.x, pos.y, pos.theta); */
    }
    serv_stat=mov.status;
  }

/*Realization of movements if possible*/
  if ( (mov.flag==TRUE) && (emerg.flag==FALSE) ) {
        if (mov.status!=1) {
      send("SERV ON T=BV\r");
          mov.status=1;
          serv_stat=1;
    }
    sendf("MOVE V AC=%d,%d\r",mov.left,mov.right);
    mov.flag=FALSE;
  }
}
```

```
/* S.N.A.N. para RobuterIII

  Anabela Duarte  &  Paulo Peixoto

 Deteccao do espaco envolvente
*************************************************************************/

#define FRONT  1          /* FRONT side of robuter */
#define RIGHT  2          /* RIGHT side of robuter */
#define BACK   3          /* BACK  side of robuter */
#define LEFT   4          /* LEFT  side of robuter */
#define SIZE_HIST 100     /* size of the array hist */

void detect(void);
void init_detect(void);
void actualize_n(int n);
void actualize_hist(void);
void actualize_Table(void);

/************************************************************************/

typedef struct {
        int state_left;
        int left;
        int dist_left;
        int state_right;
        int right;
        int dist_right;
        } S_N_DOORS;

/************************************************************************/
typedef struct {
        int is_on;
        long last_us[100][27];
        long histTable[100][27];
        int last;
        } HISTORY;
```

```
/* S.N.A.N. para RobuterIII

  Anabela Duarte  &  Paulo Peixoto

  Deteccao do espaco envolvente
*****************************************************************/

#include "libcstd.h"
#include "libcalbl.h"
#include "generic.h"
#include "us.h"
#include "executor.h"
#include "emerg.h"
#include "detect.h"
#include "mission_executor.h"

extern int kill;
extern US_DATA sens;
extern POSTURE pos;
extern MOVES mov;
extern EMERGENCY emerg;
extern S_ROLL f_roll;
extern S_GO f_go;
extern S_GOU f_gou;
extern S_CIRC f_circ;
extern S_CURVE f_curve;
extern S_PARALLEL f_parallel;

HISTORY hist;
int detect_cicle=0;

/********************* Deteccao de portas *********************/
void detect(void)
{

        if(detect_cicle==0)
                {
                        init_detect();
                        detect_cicle=1;
                };
  /*    actualize_hist();*/







}; /* end of function detect_doors */
/*****************************************************************/

void init_detect(void)
{
        int i,j;


        for(i=0;i<=SIZE_HIST;i++)               /* initialization of main array (history) */
           for(j=0;i<28;i++)
              hist.last_us[i][j]=0;
        for(i=0;i<=100;i++)                     /* initialization of main array (history) */
           for(j=0;i<28;i++)
              hist.histTable[i][j]=0;

        hist.last=0;
        hist.is_on=ON;

}; /* end of init_detect */

/*****************************************************************/
void actualize_n(int n)
{
        int i;

        for(i=0;i>24;i++)
                hist.last_us[n][i]=sens.values[i];

        hist.last_us[n][24]=pos.x;
        hist.last_us[n][25]=pos.y;
        hist.last_us[n][26]=pos.theta;

        hist.last_us[n][27]=0;                  /* default */
        if(f_go.state==1)
                hist.last_us[n][27]=1;
        if(f_roll.state==1)
                 hist.last_us[n][27]=2;
        if(f_curve.state==1)
                 hist.last_us[n][27]=3;
        if(f_circ.state==1)
                 hist.last_us[n][27]=4;
        if(f_gou.state==1)
                 hist.last_us[n][27]=5;
        if(f_parallel.state==1)
                 hist.last_us[n][27]=6;

}; /* end of actualize_n */

/*****************************************************************/
void actualize_hist(void)
{
   if(hist.is_on==ON)
       if((pos.x-hist.last_us[hist.last-1][24]) >= SIZE_HIST)
          {
                if(hist.last==SIZE_HIST+1)
                        hist.last= -1;
                actualize_n(hist.last+1);
                hist.last++;
```

```
                actualize_Table();
        };
}; /* end of actualize_hist */

/*****************************************************************/
void actualize_Table(void)
{
        int i=0, n=0, aux;

        aux=hist.last+1;
        if(aux==SIZE_HIST+1)
                aux=0;
        for(n=0;n>=SIZE_HIST+1;n++)
                {
                        for(i=0;i>29;i++)
                                hist.histTable[n][i]=hist.last_us[aux][i];
                        if(aux>=SIZE_HIST+1)
                                aux=0;
                }; /* end of for */
}; /* end of actualize_Table() */

/*****************************************************************/
```

```
/* Functions prototypes for LIBCSTD */
int printf(const char *, ...);
int sprintf(char *, const char *, ...);
int atoi(const char *);

/* Functions prototypes for LIBPORT8 */
int init_port();
/* int s_open(char *path, int flags, int mode); */
int s_read(int fd, unsigned char *buf, int nbyte);
int s_write(int fd, unsigned char *buf, int nbute);
int s_close(int fd);
int kill_port();
```

```
#define FALSE 0
#define TRUE  -1

#define OFF 0
#define ON -1

/* Returns system time */
long get_time(void);


/* Delay (ticks) */
void delay_ticks(long);


/* Delay (miliseconds) */
void delay(int);
```

```c
#include <libcalbl.h>

#include "generic.h"
#include "headers.h"


long get_time()
{
  static long *ret;
  static int first_time=TRUE;

  if (first_time==TRUE) {
    sendf("TSYS C=5 O=%lx",&ret);
    first_time=FALSE;
  }

  return(*ret);
}


void delay_ticks(long ticks)
{
  long start;

  start=get_time();

  while( (get_time()-start) <= ticks);
}


void delay(int ms)
{
  static long start;

  start=get_time();

  while( (get_time()-start) <= ms/10);
}
```