Instituto Superior Técnico

Licenciatura em Engenharia Informática e de Computadores

Projecto Final de Curso 98/99

Interface Gráfica para Operação de Robots Cooperantes

Bruno Miguel Correia, nº 41332
Sérgio Rodrigues Saraiva, nº 41440

Professor Orientador: Pedro Urbano Lima

RESUMO

O trabalho final de curso desenvolvido integra-se no projecto *SocRob* (*Soccer Robots* ou *Society of Robots*), em desenvolvimento no ISR/IST (Instituto de Sistemas e Robótica/Instituto Superior Técnico). O trabalho consiste numa **interface gráfica** para operação (**tele-controlo** e **telemetria)** de uma **comunidade de robots futebolistas**, construídos para participar em eventos internacionais da especialidade. No entanto, algumas funcionalidades do trabalho foram implementadas para operações de uso genérico e não apenas no âmbito do futebol robótico.

Palavras chave: interface gráfica, tele-controlo, telemetria, comunidade de robots futebolistas.

ÍNDICE

ÍNDICE DE IMAGENS

1. INTRODUÇÃO

No âmbito da iniciativa *RoboCup* [1], o futebol robótico tem como objectivo desenvolver uma equipa de humanóides completamente autónomos capazes de vencer a equipa humana vencedora do campeonato do mundo do ano de 2050. Apesar desta forte componente de entretenimento e diversão, o conhecimento adquirido pela investigação nesta área pode ser utilizado noutros domínios, como por exemplo, o salvamento em desastres de grande escala, tais como os terramotos. A semelhança nestes dois domínios prende-se sobretudo por ambos possuírem ambientes dinâmicos, informação incompleta e com ruído.

Este trabalho consistiu no desenvolvimento de uma interface gráfica para operação de uma comunidade de robots. O trabalho esteve integrado no âmbito de competições de futebol robótico, no qual os alunos envolvidos participaram em conjunto com a restante equipa (*ISocRob* [2][3][4][5][6]) no campeonato do mundo de futebol robótico, *RoboCup99* [1], realizado em Estocolmo de 26 de Julho a 4 de Agosto de 1999, tendo ficado na 9º posição entre 20 equipas participantes.

A aplicação principal desenvolvida consiste numa peça de *software* que através de uma interface gráfica permite efectuar a observação do estado dos robots através da telemetria e lançamento remoto de um jogo através especificação dos parâmetros de jogo como sendo jogador: guarda-redes ou jogador avançado, cor da baliza, etc. Foi também efectuada a integração de um robot comercial *Super Scout II* na equipa existente, que consistiu na alteração física da plataforma e transporte do software existente para este novo robot.

Existe um conjunto de funcionalidades não directamente relacionadas com o futebol robótico, como sendo o controlo manual dos robots, permitindo controlar remotamente o andamento de um robot com a ajuda de um periférico *space mouse*, ou com um conjunto de botões contidos na interface gráfica existentes para o efeito. Também é possível visualizar a imagem captada em tempo real pelas câmaras instaladas em cada robot. Estas duas funcionalidades usadas em conjunto podem por exemplo transformar a comunidade de robots futebolistas, numa comunidade de robots "exploradores", não completamente autónomos e controlados remotamente, permitindo explorar terrenos hostis.

Por fim, é também possível através da aplicação editar, gravar e activar parâmetros internos de cada robot, como sendo a aceleração e os controladores dos motores, as variáveis internas dos robots que se desejam visualizar por telemetria.

A maioria dos módulos foram desenvolvidos de uma forma independente e apenas integrados posteriormente na interface gráfica. Assim, é possível executar módulos independentemente e separadamente do resto da aplicação. Para além disso é também relativamente fácil transportar os módulos para outras aplicações.

A aplicação foi desenvolvida no sistema operativo *Linux* [7], distribuição *SuSe*. As soluções propostas e apresentadas dizem respeito a este ambiente de

desenvolvimento. A linguagem de programação usada, foi a linguagem C onde se integrou a linguagem *script* Tcl/Tk.

2. FERRAMENTAS E CONCEITOS UTILIZADOS

Neste capítulo serão apresentados alguns dos conceitos utilizados no trabalho. Esses conceitos serão expostos com relativo grau de detalhe porque apesar de serem os conceitos base do trabalho, não é intenção dos autores uma descrição exaustiva dos mesmos. Além disso, os conceitos serão sempre apresentados dentro do contexto do trabalho desenvolvido.

2.1. Tarefas (*threads*)

Uma das capacidades dos sistemas operativos actuais é a possibilidade de execução de vários processos em simultâneo (*multitasking*). Começa a ser comum o desenvolvimento de programas compostos por um conjunto de sub-tarefas. Até agora, apenas existia uma forma de dividir um programa em várias tarefas: criando novos processos. Mais recentemente apareceu um alternativa que é muito mais eficiente e que fornece um melhor desempenho global para as aplicações, as tarefas (*threads*) [8]. No modelo das tarefas, múltiplas execuções responsáveis por determinados fluxos de dados coexistem dentro do mesmo processo. Neste modelo um processo divide-se em duas partes: uma parte que contêm os recursos usados por todo o programa, como as instruções de código e os dados globais; uma outra que contem a informação relacionada com o estado de execução, como o contador de programa (*program counter*) e a pilha (*stack*). Esta parte também é designada de tarefa (*thread*).

2.1.1. Tarefas *versus* Processos

Em cada um dos modelos (processos ou tarefas) podemos ter programação concorrente, que fornecem vários mecanismos de sincronização, tais como, *mutexes* (semáforos de exclusão mútua), semáforos, secções críticas, etc. Dado isto, porquê a utilização de tarefas?

Criar um novo processo pode tornar-se muito ineficiente: é preciso memória (todo o processo tem que ser replicado, nomeadamente a informação contida nas páginas de memória), todas as comunicações entre processos, sincronismo a dados partilhados e outras funções envolvem chamadas ao núcleo do sistema operativo, e portanto a sua utilização não é nem trivial nem sempre eficiente.

As tarefas oferecem uma melhor alternativa uma vez que são criadas sem replicar todo o processo, o mecanismo de sincronização é feito simplesmente pela monitorização de uma variável, ou seja, as tarefas permanecem no mesmo espaço de endereçamento do núcleo do programa e por conseguinte tornam a sua utilização bastante simples e fácil. Evitam chamadas ao núcleo do sistema operativo libertando-o para outras funcionalidades.

Desta forma, a vantagem de se implementar o mecanismo de tarefas interiores a um processo reside fundamentalmente na simplicidade. Nesta solução, ao contrário da solução de criar novos processos, não é necessário gerir nem um espaço de memória partilhada, nem recursos de sincronização (semáforos, etc.) partilhados entre processos, uma vez que cada tarefa partilha o espaço de endereçamento do processo que as lançou, e tem acesso aos recursos reservados pelo mesmo. A grande desvantagem das tarefas tem a ver com o facto de sendo estas interiores a um processo, um comportamento irregular de uma delas pode ter repercusões negativas no comportamento das restantes, apesar de teoricamente serem fluxos de execução independentes.

No entanto, a solução de se usarem processos integrados com um mecanismo de memória partilhada, e não simplesmente tarefas, foi a nossa primeira (mas não definitiva) implementação avançada para os problemas que requeriam processamento paralelo com partilha de dados dentro da mesma máquina. A partir dessas implementações foram tiradas algumas conclusões.

A gestão de um espaço de memória partilhada entre processos nem sempre é trivial, uma vez que são necessárias bastantes precauções para o mecanismo ser executado de uma forma correcta, ordeira e eficiente. Desta forma, há que garantir um conjunto de condições para o efeito:

1. Nunca pode permanecer um espaço de memória reservado quando ambos os processos terminam a sua execução. Esta condição é válida tanto no caso em que os processos terminam de forma ordeira, mas especialmente quando isso não acontece. Caso esta condição não seja satisfeita, poderá ficar reservada memória que nunca será libertada. Esta condição é garantida, libertando sempre do espaço do endereçamento do processo a zona de memória partilhada quando se está em fase de terminação da aplicação (o sistema operativo encarrega-se de libertar a memória quando o ultimo processo que partilhava a mesma envia a ordem nesse sentido). Além disso, para as situações em que a aplicação pode terminar de forma abruta, é também necessário interceptar as rotinas associadas aos *signals* que podem originar uma terminação desse género. Assim, substituem-se essas rotinas por outras, que libertam do processo o espaço de memória partilhada reservado, antes de terminarem a aplicação.

2. Eficiência no que diz respeito a recursos utilizados, caso este seja um factor crítico. Por exemplo, caso se reservasse uma zona de memória partilhada para troca de poucas variáveis de controlo entre os processos (duas variáveis), sendo que o conjunto dessas duas variáveis ocupava em termos físicos 8 *bytes*, então o sistema operativo reservaria uma página de memória para apenas serem usados 8 *bytes* dos muitos *Kbytes* que uma página dispõem (em *Linux* não é possível reservar um número fixo qualquer de *bytes*: o sistema operativo apenas reserva páginas de memória, logo reservará tantas páginas quantas forem necessárias para suportar a quantidade de memória requerida pelo utilizador).

3. O sincronismo para acesso à memória partilhada terá também de usar recursos partilhados do sistema operativo entre os processos. Existem algumas bibliotecas que implementam primitivas de sincronismo. Dependendo da biblioteca

que se utilizar são possíveis várias soluções. Em algumas bibliotecas, cada recurso (neste caso semáforos) contém um identificador universal dentro da máquina, logo os processos apenas terão de conhecer à partida esse identificador (com a memória partilhada acontece o mesmo, e de certa forma algo idêntico também com a comunicação entre processos via *sockets*, uma vez que para um processo enviar um enviar um pacote para outro, ambos terão de conhecer o número de porto dentro da máquina utilizada para recepção). Noutras bibliotecas, para se implementar um semáforo partilhado, terá primeiro de se reservar um espaço de memória partilhada, e de seguida declarar o semáforos dentro desse espaço de memória. Mais uma vez, reservar uma página de memória para a preencher apenas com a informação relativa a um semáforo pode corresponder a um grande desperdício de recursos. No entanto e no caso deste projecto, foi usada uma biblioteca que implementa a solução dos identificadores universais dentro de cada máquina.

No início da implementação, usou-se processos como tecnologia fundamental de desenvolvimento dos problemas que necessitavam de paralelismo. Mas, pela experimentação e aprendizagem das *threads* chegou-se à conclusão (pelas vantagens do uso de threads face aos processos) que a sua implementação é mais fácil, e apresenta um melhor desempenho para o sistema global.

Para consultar um exemplo da criação de dois processos recorrendo à primitiva *fork* ver apêndice A, secção A1. Para o caso das tarefas, consultar o exemplo do apêndice A, secção A2.

2.2. A Linguagem Tcl/Tk

Tcl é a abreviatura de *Tool Command Language*. A linguagem foi desenvolvida por John Ousterhout, e destinava-se inicialmente a ser integrada em aplicações de linguagem compilada [9][10][11][12][13]. Cedo se tornou uma linguagem de uso geral, especialmente para interfaces gráficas e aplicações existentes. O Tcl é actualmente munido de duas funcionalidades: uma linguagem de *scripts* e uma linguagem de extensão de aplicações que consiste num interpretador para a aplicação que vai ser embebida na nova aplicação a realizar. É fácil fazer uma aplicação usando apenas Tcl e Tk, mas isso pode não ser desejável para aplicações que dependam de cálculos pesados, devido à lentidão inerente a este tipo de linguagens ditas de interpretadas. Este tipo de linguagens são sobretudo utilizadas como *scripting* e *front-ends* para outras aplicações.

O uso do Tcl/Tk numa aplicação poder ser feito de três modos:

1. A aplicação é toda construída em Tcl, eventualmente chamando comandos do sistema, tal como uma aplicação *shell script*. Como uma linguagem de *scripts*, o Tcl é similar a outras linguagens como o Bourne Shell.

2. O motor da aplicação: a linguagem na qual os comandos de alto nível são executados, é uma linguagem compilada, sendo o Tcl acedido através de ficheiros de configuração e de comandos de aplicação, o que é ideal para configurar aplicações.

3. O núcleo da aplicação é em C (ou outra linguagem compilada) e o motor da aplicação é em Tcl/Tk (modo utilizado neste projecto).

Nos modos 2 e 3 é utilizado o Tcl como linguagem de extensão de aplicações, existem no entanto outras linguagens que permitem efectuar essa extensão como por exemplo o VB, Perl, Python entre outras. No entanto o Tcl é mais vocacionado para ser utilizado em integração com a linguagem C, uma vez que o próprio Tcl foi desenvolvido nessa linguagem. Mas a grande vantagem do Tcl é a sua extensão gráfica: o Tk, que permite, de um modo simples e rápido construir interfaces gráficas poderosas.

O pacote gráfico Tk apresenta as seguintes vantagens:

1. O desenvolvimento de interfaces gráficas é rápido porque, uma vez que a linguagem é interpretada não é preciso esperar que a aplicação compile cada vez que se executa uma alteração na interface gráfica. Esta funcionalidade é muito importante porque grande parte da aplicação é desenvolvida de uma forma iterativa, onde os objectos vão sendo expostos no ecrã e desta forma vai-se sucessivamente verificando como evolui o resultado final.

2. Permite a dissociação com a aplicação principal.

3. Está munido de um enorme conjunto de objectos gráficos (botões, caixas, menus, etc.) suficientes para as necessidades normais de uma aplicação gráfica exigente.

O Tcl/Tk fornece ainda uma máquina virtual que torna a aplicação portável para qualquer plataforma.

Resumindo, o Tcl/Tk permite estender graficamente uma aplicação de uma maneira fácil, estável e robusta.

2.3. Definições e Atributos de Agentes e Micro-Agentes

O conceito de agente é bastante vago pelo facto de, antes de mais ser um paradigma e de apresentar variadas áreas de aplicação o que contribui para a impossibilidade de um consenso geral em relação a uma definição única. Numa perspectiva do futebol robótico, um agente é um sistema computacional, que para além das características básicas de autonomia, persistência e sociabilidade, deve também ser conotado com atributos reconhecidos a um ser humano.

Uma das propostas mais abrangentes é a de Wooldridge e Jennings [14] [15], que não é uma verdadeira definição na acepção do termo, mas sim um conjunto de atributos que os agentes devem possuir, os quais se dividem em dois grupos: o de atributos essenciais e o de atributos opcionais.

Deste modo e no enquadramento do futebol robótico um agente (robot) deverá possuir no mínimo os seguintes atributos essenciais:

1. Autonomia: os agentes operam sem a intervenção directa dos seus utilizadores, e têm informação sobre o seu estado interno assim como algum tipo de controlo sobre as suas acções.

2. Mobilidade: a capacidade de um agente se mover dentro do campo de jogo e ser capaz de realizar as suas tarefas e de cumprir os seus objectivos.

3. Sociabilidade: os agentes interagem com outros agentes e possivelmente com um utilizador (por telecontrolo ou telemetria).

4. Reactividade: os agentes analisam o seu ambiente e respondem às alterações nele ocorridas, normalmente são compostos por quatro componentes: percepção, processamento, acção e comunicação. Cada um destes componentes é designado por micro-agentes. Os micro-agentes são entidades independentes responsáveis por uma tarefa, que juntos formam um agente.

Na figura abaixo é ilustrada a forma como os micro-agentes interagem entre si, com os outros agentes e com o ambiente que os rodeia. Ao ciclo de micro-agentes percepção, processamento e acção é também comum designar-se por *STA loop* (*sense*, *think* e *act*).

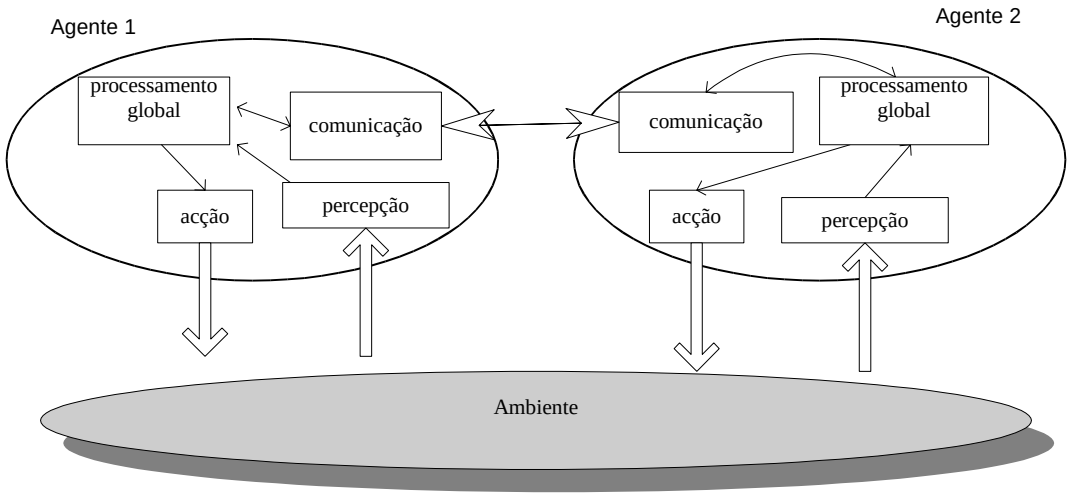


Figura 2.1: Esquema da relação entre agentes, micro-agentes e o ambiente que os rodeia.

Os atributos considerados opcionais, são tipicamente:

1. Veracidade: A característica de um agente em não comunicar informação falsa, quer seja aos agentes com que interagem quer com o próprio utilizador.

2. Benevolência: Um agente é benevolente se aceita os objectivos de outros, para o caso em que seja solicitado.

3. Características mentais: Modelo de um agente baseado em noções mentais, tais como o conhecimento, emoções, desejos e intenções. Este tipo de modelo é uma área importante no seio da comunidade actual da inteligência artificial.

Na finalização deste projecto, a comunidade existente já estava dotada de todos os atributos considerados essenciais, à excepção do atributo da sociabilidade que ainda não apresentava uma robustez necessária para essa atribuição.

Como evolução natural da comunidade, consideramos que numa fase inicial deve ser dado mais ênfase aos atributos essenciais, tornando-os robustos e fiáveis, antes de se avançar para os outros atributos ditos de opcionais, mas não menos importantes.

2.4. *Sockets*

A versão de *Berkeley* do sistema operativo *Unix* introduziu em 1982 uma proposta inovadora de interface de comunicação designada por *sockets* [16]. Os *sockets* propunham-se dar resposta a alguns requisitos:

1. Independência do protocolo por parte da interface.

2. Transparência da comunicação, devendo esta ser independente da localização física dos processos, e logo idêntica para o caso local ou distribuído.

3. Compatibilidade, o novo mecanismo deveria inserir-se na interface clássica de comunicação de entradas e saídas do *Unix*.

Os *sockets* apresentam assim uma interface baseada em descritores de ficheiros e foram dotados de independência entre o modelo de comunicação e o protocolo de transporte. Esta característica materializou-se na noção de domínio. Especificamente, no nosso caso é utilizado o domínio *internet* que utiliza os protocolos TCP (*transmission control protocol*) e UDP (*unit datagram protocol*). O primeiro protocolo (TCP) fornece um serviço de transporte com ligação (também designado por socket *stream*) que oferece um canal de transmissão fiável, bidireccional e com garantia de sequencialidade. A interface é do tipo sequência de octetos. Devido a estas características de fiabilidade, este protocolo é usado em praticamente todas as comunicações da nossa aplicação, excepto na transmissão de imagem que utiliza o protocolo UDP. Este protocolo oferece um serviço bidireccional, sem ligação (também designado por socket *datagram*) não sendo por este garantida a sequencialidade, fiabilidade ou eliminação de pacotes duplicados.

Para consultar um exemplo de utilização de *sockets* para comunicação entre dois processos *Linux*, ver apêndice A, secção A5.

3. EVOLUÇÃO DO PROJECTO

Neste capítulo descreve-se o ponto em que os alunos encontraram o projecto, as metodologias que aplicaram para a implementação das soluções dos problemas propostos, e os estados por que passaram durante essa mesma implementação e as conclusões tiradas.

3.1. Metodologias Aplicadas

O projecto apresentava para os alunos características de projecto inovador. Sendo assim, foi necessário estudar algumas áreas que não eram de total domínio para os mesmos, nomeadamente:

1. Algumas características muito especificas do sistema operativo utilizado (*Linux* distribuição *SuSe*), que sendo genericamente do conhecimento dos alunos, não era por estes completamente dominadas. Neste contexto insere-se o mecanismo das tarefas (*threads*) da biblioteca *pthreads* do *Linux*.

2. O *driver* de captação da imagem da câmara dos robots, onde foi utilizado um pacote de *software* disponível na *internet*, mas que no entanto teve de ser integrado na nossa aplicação.

3. A linguagem *script* Tcl/Tk para o desenvolvimento de aplicações gráficas no ambiente gráfico do *Linux* teve de ser aprendida e dominada, como também alguns detalhes da biblioteca gráfica *X-Windows* [17].

4. A arquitectura dos robots já existentes (motores, câmara, modems) foram também alvo de análise e de testes exaustivos.

O método utilizado para compreender e experimentar o desenvolvimento destas tecnologias foi obviamente o da leitura de documentação disponível, quando necessário pesquisada na *internet*, seguido da implementação de pequenos programas (alguns disponíveis em apêndice) para testar as funcionalidades pretendidas.

Para consultar pequenos programas de teste implementados durante o projecto para testar funcionalidades requeridas consultar apêndice A.

Noutros casos, como no mecanismo de transmissão da imagem, primeiro foi implementada uma solução que recorria a uma tecnologia já largamente dominada (processos), e que de seguida foi evoluída para uma solução mais interessante (tarefas). Poderá parecer uma táctica pouco correcta para o desenvolvimento de *software*, uma vez que seria aconselhável desenvolver a solução ideal de raiz. No entanto, por uma questão de análise de riscos optou-se por vezes em desenvolver soluções que tecnicamente pudessem não ser as ideais, mas cujas tecnologias eram completamente dominadas e assim tornava-se muito mais fácil fazer previsões

temporais do tempo de desenvolvimento, bem como garantir a concretização das soluções propostas. Além disso, sendo os módulos do trabalho independentes, uma reengenharia de um dos módulos não teria repercussões noutros módulos. Novamente no caso da aplicação de transmissão da imagem, mas também no caso da aplicação que lê as variáveis internas do robot e as posiciona numa janela de *X-Windows*, apesar de ambas estarem todas integradas na interface gráfica, foram inicialmente desenvolvidas como aplicações independentes que de seguida foram integradas na mesma interface. Tipicamente, essa integração não foi muito mais que transformar o código das aplicações em bibliotecas construindo para o efeito um ficheiro de cabeçalhos e renomeando a função principal (*main*) para uma função que a interface gráfica invoca quando é activada a opção que dispara o arranque da funcionalidade. Dada a independência dos módulos, as aplicações primitivas podem ser disponibilizadas caso se queira por exemplo ter uma aplicação que apenas transmita imagem entre computadores, etc.

A metodologia utilizada foi assim tipicamente uma metodologia em espiral[18], onde se construiu um protótipo da interface gráfica, que foi sendo sucessivamente refinado com as funcionalidades disponíveis, à medida que estas iam sendo implementadas em paralelo e de uma forma independente.

3.2. Estados de evolução da interface gráfica

Este projecto, como já foi afirmado, teve características (para os alunos) de um projecto inovador, numa área ainda pouco explorada. Dessa forma não foi trivial apresentar uma especificação rígida e definitiva antes de se iniciar a implementação dos objectivos. Inicialmente foi apresentado um relatório preliminar com os objectivos propostos, e pouco depois iniciou-se a fase de implementação. No entanto, e uma vez mais dada a pouca experiência por parte dos alunos em projectos desta natureza muitas das ideias iniciais acabaram por ser postas de parte, uma vez que apesar de teoricamente parecerem interessantes, na prática revelaram-se pouco eficazes.

Exemplo disso é a arquitectura inicial da interface gráfica que seguia uma filosofia relativamente diferente da arquitectura final. Essa primeira arquitectura era orientada à entidade existente, ou seja, o agente: assim todo o contexto da aplicação se centralizava em torno desta. A aplicação executava um protocolo de procura de agentes activos (*broadcast* de uma mensagem de estabelecimento de comunicação) e de seguida para cada agente do qual obteve resposta criava uma janela com as características base do agente: nome, endereço na rede, especialidade do agente (guarda-redes, avançado) etc. Era também facultada para cada robot a hipótese de executar um *zoom* para as suas características específicas. Essas características dividiam-se em hierarquia (motores, imagem, comunicações, etc.). A arquitectura estava pensada para ser muito facilmente refinada com novas hierarquias. Cada uma dessas hierarquias estava representada por uma opção de um menu e cada uma dessas opções evoluía para uma janela sendo que todas as janelas eram completamente independentes entre si, a fim de garantir facilmente a remoção ou inserção de novas

hierarquias. Cada robot tinha por defeito as hierarquias motores, imagem, comunicações e sensores, e cada instância de agente herdava essas hierarquias.

Na fase de análise, esta opção pareceu ser a mais lógica e evoluída: tínhamos uma aplicação que se centrava em torno da entidade agente, e não nas funcionalidades oferecidas pelos mesmos. Cada agente herdaria por defeito um conjunto de características que seriam as características genéricas da maioria da comunidade. O que fosse particular do agente, teria de ser modificado e/ou acrescentado manualmente. Fazendo um paralelismo com os paradigmas de programação, poder-se-ia dizer que era uma abordagem orientada ao objecto, neste caso ao objecto agente. Cada agente possuía por exemplo motores, cada motor é novamente um objecto com as suas características próprias (controladores, aceleração, etc.) que são assim herdadas do objecto motores já definido por defeito. Para os agentes com motores particulares, este objecto teria de ser implementado e inserido na aplicação manualmente. Este processo terá sempre de se resumir a sobrepor a referência do antigo objecto motores pela particular.

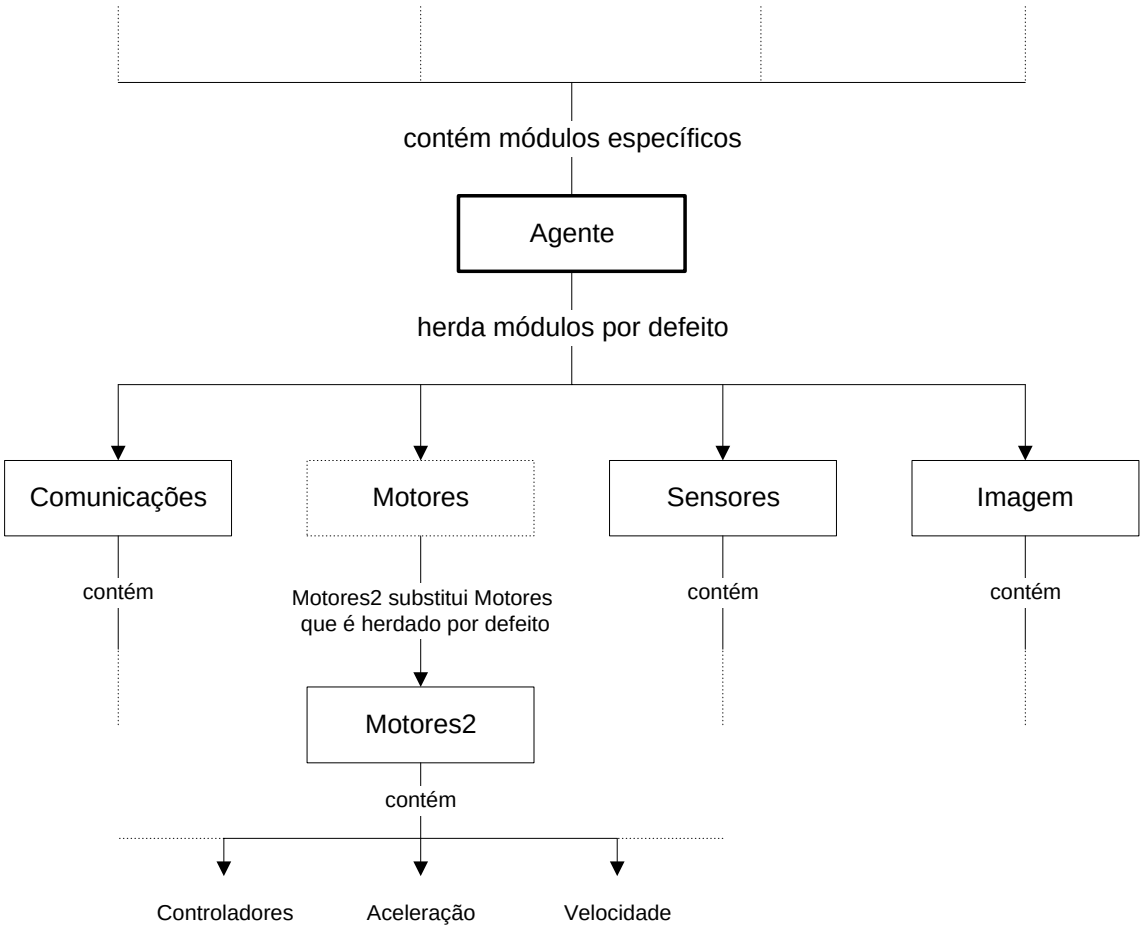


Figura 3.1: Exemplo da arquitectura inicialmente proposta, orientada à entidade agente.

Numa abordagem às funcionalidades temos pelo contrário um conjunto de operações seleccionáveis: ver imagem, controlar manualmente, ver estado interno, etc. Após a escolha da operação pretendida identifica-se o robot no qual se vai executar a essa mesma operação. Neste caso entramos num universo procedimental: existe um

conjunto de procedimentos e para executar cada um deles é necessário passar-lhe os argumentos respectivos (neste caso o robot no qual se pretende executar a funcionalidade seleccionada).

Teoricamente a opção imediata é a primeira. Mas à medida que se foi evoluindo na implementação da aplicação tornaram-se óbvios uma série de um problemas técnicos.

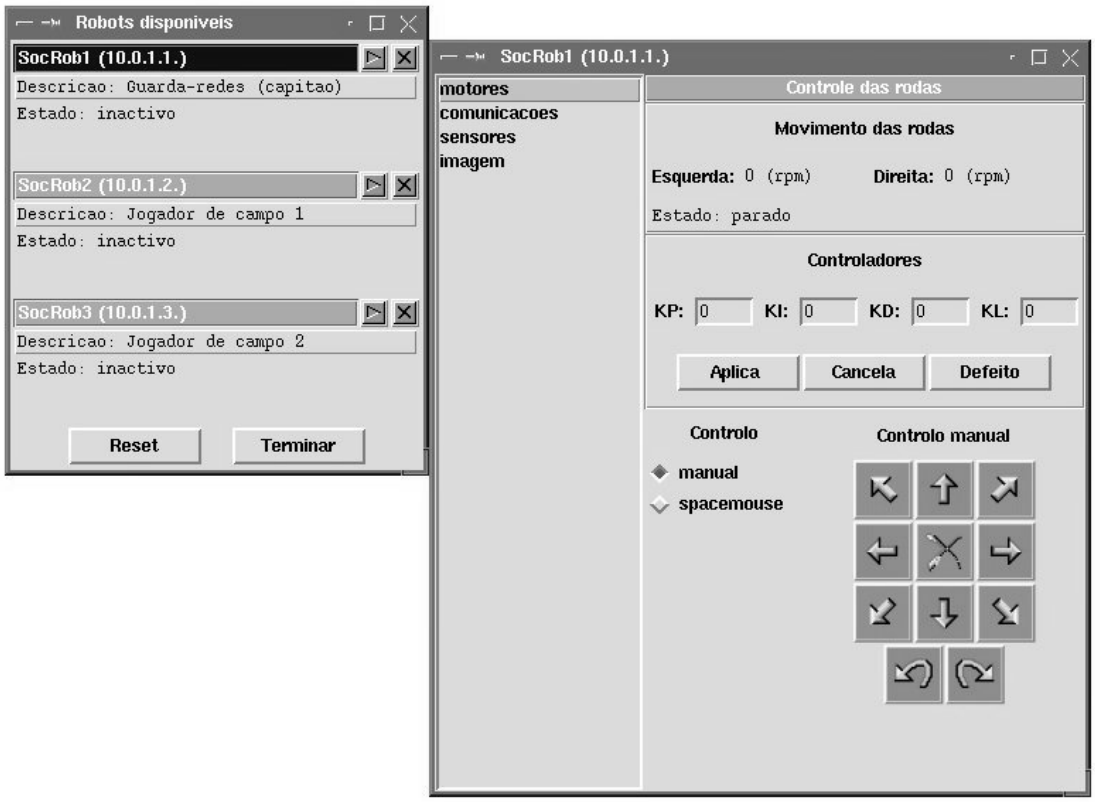


Figura 3.2: exemplo do aspecto gráfico da arquitectura inicialmente proposta (ainda um protótipo sem funcionalidades).

A linguagem utilizada para codificar a interface gráfica o Tcl/Tk (que é um linguagem interpretada bastante primitiva), contém um conjunto extenso de limitações. Nomeadamente ao nível da declaração de tipos compostos. Ou seja, não foi nada trivial implementar uma solução desenhada para ser orientada aos objectos, com heranças de propriedades, e polimorfismo, numa linguagem cujo único tipo base é o tipo cadeia de caracteres (*string*). Note-se que o objectivo do Tcl/Tk não é mais que facilitar a implementação de aplicações muito simples, nomeadamente e para o nosso caso de uma interface gráfica para o ambiente gráfico do *Linux*. No entanto não é trivial, nem é esta a ferramenta ideal para implementar aplicações com relativo grau de evolução, como seria o caso. A solução seria a de não usar o Tcl/Tk de uma forma interpretada, mas usar directamente as primitivas fornecidas pela biblioteca de Tcl/Tk no código C da aplicação. Aí, dentro da linguagem C, já seria possível declarar as estruturas de dados complexas que não são possíveis de especificar no Tcl/Tk. Mas nesse caso, perderiam-se todas as vantagens (que são muitas) de usar o interpretador

de Tcl/Tk, e entraria-se quase numa solução de desenvolver uma interface gráfica em ambiente *X-Windows* em C, o que obrigaria a uma nova aprendizagem de uma biblioteca bastante complexa, e a abandonar totalmente a solução inicialmente avançada, sobre a qual já muito código tinha sido desenvolvido. Desta forma, entrou-se em linha de conta relativamente ao compromisso de uma implementação ideal *versus* uma implementação realizável em tempo útil. Evoluir-se assim para a segunda solução: a solução funcional, que não necessita da complexidade de estruturas da solução orientada aos objectos. Apesar disso, muito código foi reaproveitado de uma solução para outra, não tendo sido de forma nenhuma tempo perdido o ter-se começado por uma solução diferente apresentada no final, como foi o caso.

Para além disso, um esquema alternativo de menus funcionais entretanto apresentado pareceu ser uma opção mais intuitiva, visualmente mais apelativa e também mais fácil de manusear. Apesar do modelo orientado ao objecto ser tipicamente mais interessante e evoluído de representar a realidade, não implica que se use esse modelo inflexivelmente para todas as aplicações, como acabou por acontecer no nosso caso. A arquitectura da interface gráfica final é descrita exaustivamente no capítulo 4, pelo que não é aqui aprofundada.

Foram aprendidas várias lições que serão sem dúvida úteis em projectos semelhantes futuros. Sumariamente temos:

Vantagens da abordagem procedimental *versus* objectos:

1. Uma abordagem orientada ao objecto no qual a aplicação se refere é mais intuitiva de perceber. Tipicamente numa comunidade de agentes existem agentes, e logo a aplicação deve centralizar-se em torno destes.

2. A abordagem procedimental é algoritmicamente muito mais simples de desenvolver, não requerendo para o efeito estruturas de dados muito complexas.

Vantagens do Tcl/Tk *versus* biblioteca *X-Windows*:

1. Tcl/Tk é muito, mas muito mais simples de utilizar para construir interfaces gráficas que a biblioteca gráfica *X-Windows*. Tipicamente para implementar um objecto de Tcl/Tk que necessita apenas de uma instrução com uma sintaxe muito trivial, é necessário invocar cinco ou seis funções da biblioteca de *X-Windows* que por vezes necessitam de estruturas de dados nada triviais.

2. Grande parte das aplicações gráficas para *Linux* são construídas utilizando o Tcl/Tk, logo utilizando objectos gráficos semelhantes (menus, botões, etc.), o que normaliza o aspecto gráfico das aplicações, não sendo necessário assim para um utilizador reaprender a usar esses mesmos objectos.

3. Tcl/Tk é uma linguagem interpretada, e logo a sua execução é muito pouco eficiente. O X-Windows é uma biblioteca que é ligada pelo compilador de C ao código fonte da aplicação a ser compilada.

4. Sendo o *X-Windows* uma biblioteca que se liga ao código C de uma aplicação, tem de se utilizar nas funções por estas disponibilizadas estruturas de C. O C é uma linguagem muito forte ao nível das estruturas de dados, ao contrário do Tcl/Tk que só dispõem do tipo cadeia de caracteres. O C é também muitíssimo poderoso em relação a estruturas de controle, ao contrário do Tcl/Tk.

Após estas conclusões, se o projecto começasse a ser desenvolvido agora, a opção tomada seria ainda a de utilizar o Tcl/Tk, mas apenas para a construção dos objectos gráficos situando toda a algoritmia do controlo da aplicação e todas as estruturas de dados declarada em código C. Esta opção justifica-se uma vez que desta forma cada funcionalidade seria implementada na linguagem para a qual é mais vocacionada.

4. ESTRUTURA DE *SOFTWARE*

Neste capítulo é efectuada uma breve descrição da arquitectura de *software* existente em cada robot assim como o estado actual de desenvolvimento da mesma. Toda esta arquitectura foi ou está a ser desenvolvida por outros elementos da equipa *ISocRob*.

4.1. Descrição geral da arquitectura de *software* existente em cada robot

Cada robot contêm um conjunto de micro-agentes (captura de imagem, motores, máquina de estados, monitorização, etc.). Estes micro-agentes são entidades autónomas responsáveis por uma parte independente do processamento global. Todos os micro-agentes comunicam entre si escrevendo e lendo no *blackboard*.

Antes de avançarmos convêm introduzir este conceito. Conceptualmente um *blackboard* é um repositório de dados centralizado, no qual é efectuado o mapeamento de símbolos com um espaço de nomes organizado hierarquicamente e com significado (exemplo: *robot1.vision.hasball?*) para variáveis físicas. Também é possível associar acções a elementos do *blackboard* quando o seu valor é lido ou escrito.

Conceptualmente o *blackboard* é um tipo de dados centralizado, mas fisicamente é distribuído pela comunidade, ou seja, cada robot contêm a sua parte do *blackboard* que é actualizada via rede *ethernet* sem fios. Convêm no entanto referir que esta versão do *blackboard* distribuído ainda está numa fase muito prematura de desenvolvimento. Na figura abaixo é ilustrada a arquitectura de *software* de cada robot e as suas dependências.

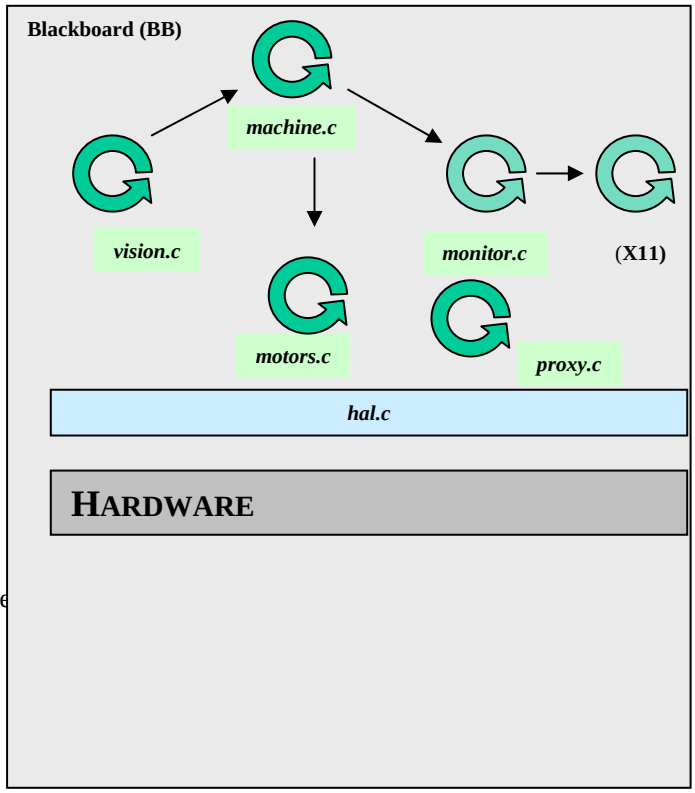


Figura 4.1: Arquite

Toda a interface com o hardware é assegurada pelo *hal* (*hardware abstraction layer*), que fornece uma camada de abstracção ao *hardware* permitindo dissociar o *hardware* do *software*.

Segue-se uma breve descrição das funcionalidades de cada um dos micro-agentes acima ilustrados:

***Vision*:** este micro-agente é responsável pela captura das imagens através de um dispositivo de visão (câmara mais placa de aquisição da imagem).

***Machine*:** cérebro do robot, contêm a maquina de estados inerente ao funcionamento global do robot.

***Motors*:** Todas as chamadas aos motores passam por este micro-agente.

***Monitor*:** efectua a monitorização do sistema, permitindo a visualização de um conjunto de variáveis do *blackboard* quase em tempo real por parte de um utilizador.

***X11*:** mostra a imagem da câmara com a detecção das cores correspondentes aos aspectos (*features*) relevantes ao jogo.

***Proxy*:** responsável pelas comunicações entre robots.

4.2. Comunicações entre Robots e a Estação de Operação

Esta secção explica as opções tomadas no que se refere à comunicação da estação de operação e os robots, e como esta se integra no restante conjunto.

4.2.1. Protocolo de comunicação

Cada pacote de trama de informação conterá a sua estrutura de dados, mas na rede apenas é reconhecido a estrutura cadeia de caracteres, ou seja, na rede passam sequências de *bytes*, sem tipo definido. Assim, foi necessário criar um par de funções: uma para ser usada do lado do emissor que recebe uma estrutura para ser enviada e a transforma numa cadeia de caracteres; a função inversa desta para ser usada do lado do receptor que recebe numa cadeia de caracteres e reconstroi a estrutura de dados original.

Para consultar um exemplo de utilização destas funções de conversão de dados, ver apêndice A, secção A6.

Desta forma cada mensagem é constituída pelo identificador da mensagem, o tamanho do corpo respectivo, e o corpo.

2 bytes	2bytes	variável
ID_MSG	TAM_CORPO	CORPO

Figura 4.2: Representação de uma mensagem.

O campo ID_MSG identifica a mensagem. Assim, cada mensagem deverá ter um identificador unívoco. Uma vez que o campo CORPO tem tamanho variável dependente da mensagem acrescentou-se o campo TAM_CORPO que contém o tamanho em *bytes* do campo CORPO. Note-se que algumas mensagens podem ter um corpo de tamanho variável. Por exemplo, uma mensagem do tipo "faixa-de-imagem" (definida no capítulo da aplicação que transmite imagem pela rede), tem um corpo de tamanho variável, logo não existe uma relação directa entre uma mensagem específica e o tamanho do seu corpo.

Os dois campos iniciais são convertidos em formato rede através das funções *htons* (*host to network short*) e *htonl* (*host do network long*) e reconvertidos para formato máquina com as funções *ntohs* (*network to host short*) e *ntohl* (*network to host long*). Desta forma garante-se a portabilidade das mensagens entre máquinas com representações internas diferentes, para a informação. Pode parecer um pouco estranho executar estas operações uma vez que estamos num ambiente em que os robots usam o mesmo tipo de processadores. No entanto, o processo de monitorização que reside numa máquina externa à comunidade não tem de ser obrigatoriamente um PC, como é o caso. Aconteceu até utilizarmos uma máquina que apresentava uma arquitectura diferente no inicio do projecto.

4.2.2. Mecanismo de *broadcast*

Para iniciar os protocolos de comunicação, o processo da interface gráfica tem de ter presente a que máquinas (robots) é possível estabelecer ligação. Para isso, consulta um ficheiro de configuração local onde residem os nomes das máquinas às quais o utilizador deseja que sejam registadas no sistema sempre que a interface gráfica é carregada. Assim, após ler este ficheiro a aplicação cria uma lista de agentes possíveis a estabelecer comunicação e começa a tentar contacta-los enviando mensagens de boas vindas para essas máquinas. Os casos para os quais for obtendo resposta vão sendo inseridos numa lista de agentes activos, e é reservada uma entrada numa uma tabela global de identificadores de *sockets*. Assim cada agente activo passa a conter um identificador unívoco interno, que representa o índice dessa tabela onde o identificador do *socket* reservado para a comunicação entre o processo da interface gráfica e o agente correspondente reside. Desta forma a dimensão dessa tabela é o limite máximo de agentes que podem estar ligados ao sistema ao mesmo tempo. No nosso caso um limite de dez agentes parece ser uma opção boa e mais do que suficiente. Finalmente para os agentes que não responderem num limite de tempo (*timeout*) razoável, assume-se que estes não estão disponíveis no momento, e são inseridos numa lista de agentes não activos.

Durante a utilização da interface gráfica, só é obviamente possível invocar operações oferecidas por esta sobre os agentes que estejam presentes na lista dos agentes activos.

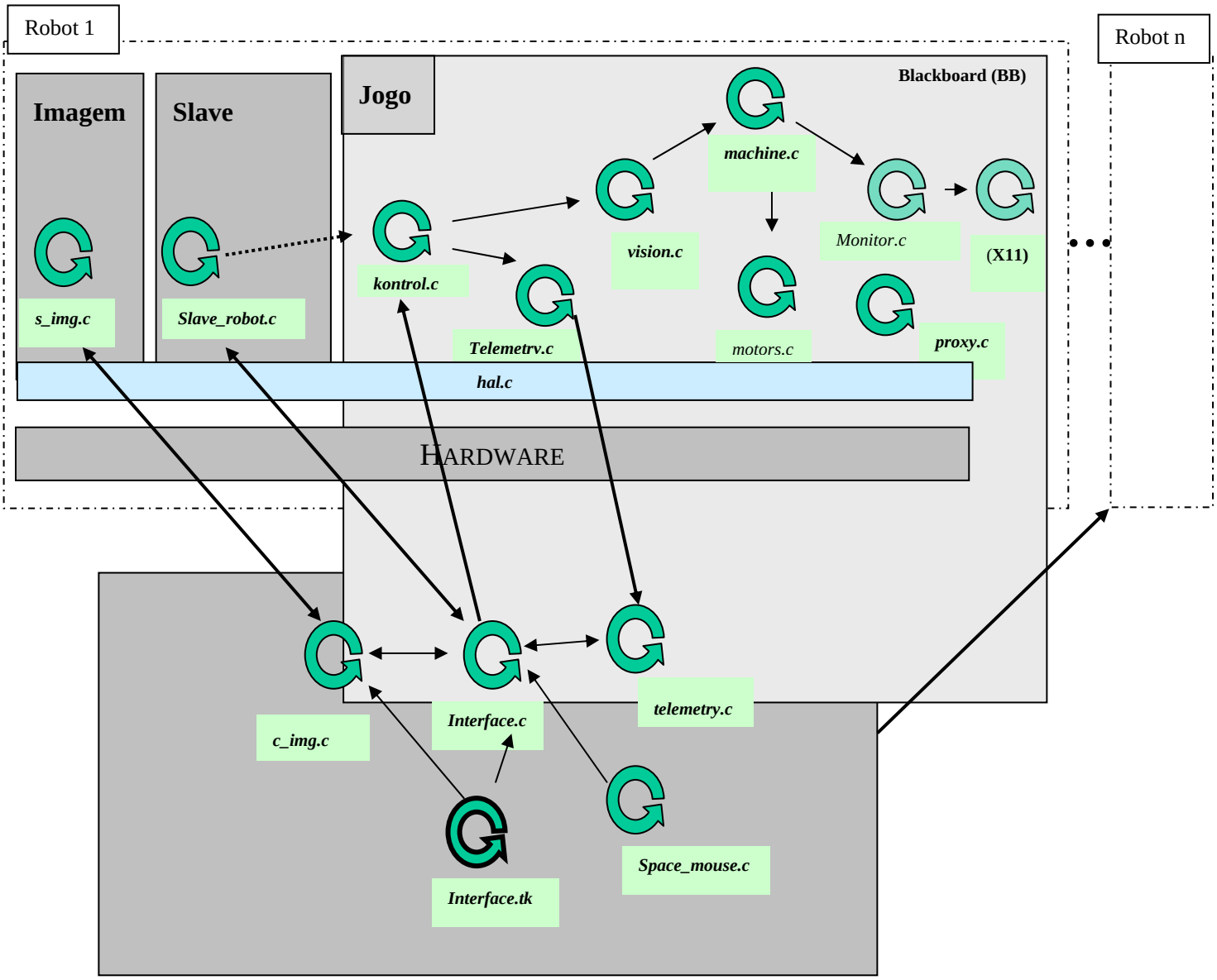
É também possível a qualquer altura, caso o utilizador deseje, voltar a correr o mecanismo de *broadcast* onde as listas de agentes referidas voltam a ser actualizadas. Acontece que durante essa reorganização interna da aplicação não é possível utilizar qualquer das funcionalidades sobre os agentes (sendo a aplicação remota aos robots todas as funcionalidades desta relativamente aos mesmos tem de ser feita através da rede) uma vez que a função de *broadcast* executa uma espécie de trinco (não no sentido exacto do contexto das primitivas de sincronismo usuais dos sistemas operativos multitarefa, mas com um sentido idêntico) sobre as comunicações.

No nosso caso existem pelo menos dois serviços distintos: um serviço baseado em *sockets* com ligação, para o caso das operações de controlo manual, lançamento automático de jogo, etc.; e um serviço baseado em *sockets* sem ligação para o mecanismo de transmissão da imagem. Os dois processos residentes do lado dos robots, mesmo que pertençam a um só processo foram pensados para serem organicamente independentes. Desta forma é possível apenas estar um serviço activo, não necessitando a aplicação de que estejam todos disponíveis para se executar. No entanto esta solução acrescenta um problema: é necessário mais que um mecanismo de *broadcast*. Neste caso concreto para o serviço de controlo e telemetria, e outro para o serviço de visualização remota da imagem. A forma mais eficiente de ultrapassar esta contrariedade foi criar duas funções de *broadcast*, uma para cada serviço, mas que são executadas em paralelo recorrendo a um mecanismo de tarefas (*threads*). Esta implementação é bastante eficiente, uma vez que estas tarefas passam a maioria do tempo bloqueadas à espera que cheguem respostas às mensagens de boas vindas enviadas, e assim é mais positivo ter duas tarefas em paralelo à espera que expire um temporizador de que ter essas esperas a serem executadas de uma forma sequencial.

4.3. Arquitectura de *software*/comunicações da aplicação desenvolvida

Nesta secção iremos descrever de uma forma suscita a arquitectura de *software* que serve de suporte a toda a aplicação desenvolvida. É efectuada a descrição das várias tarefas que compõem a aplicação principal, assim como a interacção entre estas e os robots da comunidade existente, que também é descrita.

A implementação da aplicação que consiste na interface gráfica foi desenvolvida segundo um modelo de tarefas usando *threads* da biblioteca *pthreads* do *linux* uma vez que podíamos tornar concorrentes vários fluxos de informação, de modo a optimizar os recursos do sistema. A escolha das tarefas como modelo base recaiu sobretudo pela facilidade de implementação e de possuir um conjunto de mecanismos de sincronismo simples (ver secção 2.1.), que permitem implementar com facilidade este método de programação paralelo. Seguidamente é ilustrada toda a arquitectura de *software* e de comunicações desenvolvida sobre o modelo apresentado na secção anterior.



Legenda:

Ligações com sockets

Dependências de dados e/ou tarefas

Execução através de um fork

tarefa ou processo

motor de toda a aplicação

Figura 4.3: Arquitectura de *software* e de comunicações de toda a comunidade existente (robots/utilizador)

Seguidamente é efectuada uma breve descrição de todas as tarefas que foram utilizadas no desenvolvimento da aplicação (ver figura 4.3)

Desta forma, a aplicação base da interface gráfica foi dividida nas seguintes tarefas:

Aplicação núcleo (*interface.c*)**:** tarefa que efectua todo o processamento responsável por, entre outras coisas, pela parte das comunicações entre a aplicação principal e os robots existentes, manipulação de ficheiros de configuração, etc. No fundo esta tarefa corresponde ao cérebro de toda aplicação, já que todas as tarefas directa ou indirectamente fazem referência a esta.

Space Mouse (*magellan.c*)**:** tarefa responsável pelo tratamento deste dispositivo, nomeadamente pela leitura dos valores gerados por eventos que traduzem alterações do comportamento do periférico.

Aplicação gráfica (*interface.tk*)**:** o motor da aplicação desenvolvida, ou seja, a tarefa que contêm os comandos de mais alto nível embebidos na aplicação gráfica, responsável pela actuação de todo o sistema.

Telemetria (*telemetry.c*)**:** Tarefa que é executada apenas em modo de jogo. Permite a visualização de um conjunto de variáveis pré-definidas do *blackboard* de cada robot.

Imagem (*s_img.c*)**:** a tarefa encarregue do controlo e recepção da imagem de cada robot em modo de controlo manual.

Do lado do robot coexistem as seguintes tarefas:

Slave Robot (*slave_robot.c*): tarefa responsável pela tele-operação.

Imagem (*c_img.c*): Tarefa encarregue pela captura e envio da imagem para a aplicação principal.

À arquitectura de jogo existente foram ainda acrescentadas algumas tarefas de modo a se efectuar o controlo do jogo (execução de jogos e telemetria), estas tarefas são:

Kontrol (*kontrol.c*): Tarefa responsável pela inicialização do jogo.

Telemetria(*telemetry.c*): tarefa encarregue de obter a informação pretendidada do *blackboard* e do envio da mesma para a aplicação principal.

É de notar que a tarefa *monitor.c* não faz muito sentido agora, uma vez que a tarefa telemetry.c realiza a mesma funcionalidade de uma forma melhor e mais amigável. No entanto convém não eliminar a tarefa *monitor.c* ,uma vez que esta é responsável pela execução da sub-tarefa *X11* . Esta sub-tarefa, recorde-se, mostra a imagem da câmara com a detecção das cores correspondentes aos aspectos (*features*) relevantes ao jogo, o que é bastante útil no processo de calibração das cores.

4.4 Módulos da Aplicação

Segue-se uma descrição detalhada de cada um dos módulos implementados:

4.4.1 Módulo *Slave*-Robot

Em cada um dos robots existe um processo sempre activo responsável pelo controlo manual sobre o robot em questão, assim como pela inicialização e finalização de um jogo. A relação entre este processo e a aplicação principal é baseada numa metodologia *Master-Slave*, em que a aplicação principal ordena uma acção que será realizada pelo processo *slave* existente no robot. As comunicações entre estas duas entidades são realizadas através de *sockets stream* (ver secção 2.4) pelas razões já apresentadas.

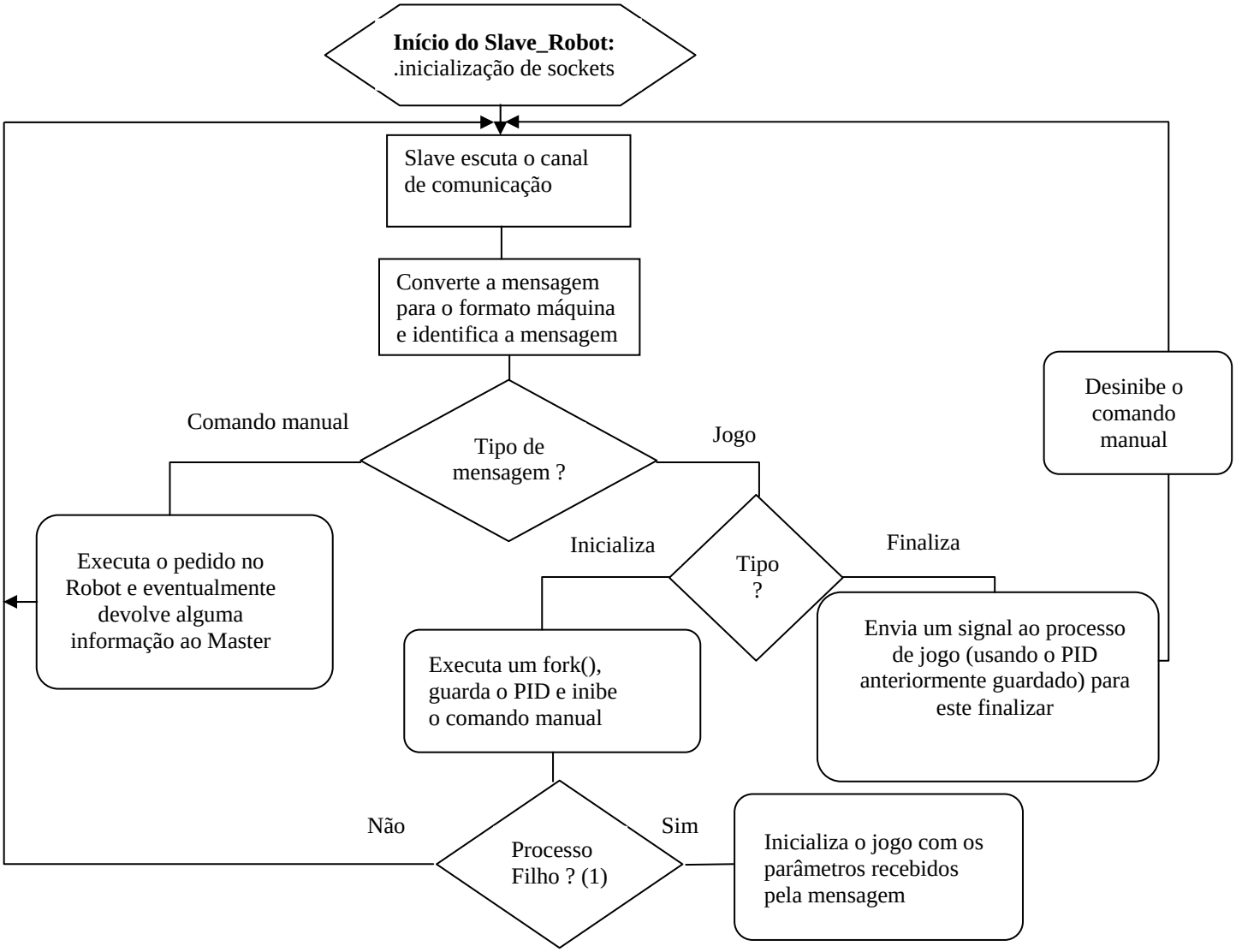


Figura 4.4: Fluxograma que traduz o funcionamento do módulo *slave*-robot

É de realçar que o *Slave*-Robot e o Jogo são programas sem nenhum relacionamento em termos de código, logo a execução do jogo é efectuada através da chamada a uma função do sistema operativo (*exec*) cuja função é executar um programa, alterando o contexto do processo actual para o contexto do novo programa, que neste caso é o jogo.

Sublinhe-se que durante o modo de jogo, o comando manual está inibido de forma a impedir o comando remoto dos robots, que não é permitido pelas regras do futebol robótico.

4.4.2 Módulos de Teleoperação

O módulo de teleoperação pode ser efectuado através do *space mouse* ou através da própria interface.

4.4.2.1 Módulo Space Mouse

Para o comando manual dos robots pensou-se inicialmente num periférico do estilo de um *joystick*. Um periférico dessa natureza apresentava-se como uma opção natural, uma vez que possui o *hardware* específico para facilitar o controlo de um qualquer objecto (tipicamente para jogos) em relação às teclas. No entanto, existia nos laboratórios um periférico *Magellan/SPACE MOUSE "Classic"*, já utilizado noutros projectos. Sendo assim, em vez de se adquirir material novo optou-se por utilizar o material existente adaptando-o para o efeito, uma vez que um *space mouse* é um periférico muito sofisticado onde as suas potencialidades vão muito mais além das requeridas para a nossa aplicação.

O *space mouse* possui um manipulador muito sensível ao toque. Esse manipulador pode ser usado para uma quantidade bastante grande de movimentos (seis graus de liberdade), nomeadamente para rotações e translações. Possui também um conjunto de botões que são utilizados na nossa aplicação, nomeadamente para mudar de robot e para controlo do robot actual.

Figura 4.5: Periférico *space mouse*

Apesar de ao início não parecer muito trivial controlar um robot com este dispositivo, depois de alguma prática torna-se bastante intuitiva e fácil a operação do

mesmo. Juntamente com a aplicação de visualização da imagem remotamente, é possível controlar um robot estando este fora do alcance de visão da pessoa que o comanda. Nomeadamente foi testada esta capacidade no ISR comandando remotamente um robot por alguns laboratórios existentes. Este tipo de funcionalidade permite deste modo a exploração de lugares hostis à presença humana.

De seguida enumeram-se as características básicas do *space mouse*:

Modos de Operação:	Interface 3D com seis graus de liberdade
Modo de Translação:	Apenas são reportadas as coordenadas de translação (X, Y, Z)
Modo de Rotação:	Apenas são reportadas as coordenadas de rotação (A, B, C)
Modo Dominante:	Apenas a coordenada com maior magnitude é reportada
Sensibilidade:	Ajustável (600 níveis de resolução de velocidade)
Botões:	Nove, programáveis
Dimensões:	L x C x A: 163 x 112 x 40 mm
Peso:	665 gr.

Este dispositivo já trazia um *driver* de suporte à plataforma *Linux*, baseado em eventos de *X-Windows*, pelo que a sua integração na aplicação foi pacífica.

Na figura ao lado direito apresentam-se os eixos de operação do periférico. Note-se que apenas se utilizam os eixos B e Z. Sendo o eixo B responsável pela rotação e o eixo Z pela translação do robot a controlar. Todos os restantes eixos são ignorados, uma vez que não têm utilidade para a aplicação em questão.

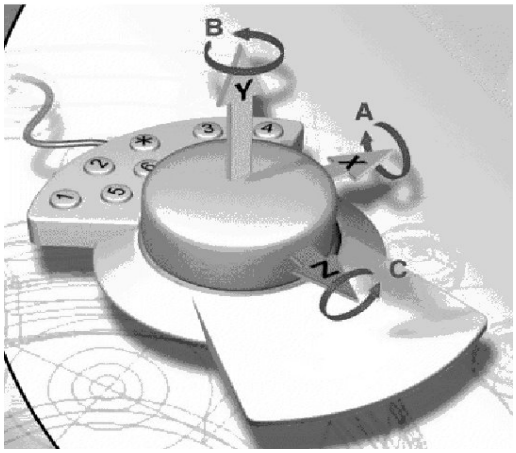


Figura 4.6: eixos de operação do *space mouse*

Desta forma, o controlo remoto de um robot pode ser efectuado manipulando o *space mouse*. Esta tarefa utiliza o núcleo da aplicação (ver figura 4.3) para comunicar com a tarefa *slave-robot* de cada robot com o objectivo de se efectuar o controlo remoto. Outro detalhe é que esta tarefa é completamente independente da aplicação gráfica, podendo exercer-se controlo sobre um robot a qualquer momento a partir da aplicação gráfica, excepto quando se faz um *broadcast* de actualização de ligações de comunicações e também quando decorre um jogo.

4.4.2.2 Módulo de comando

Através do módulo de comando (embebido na tarefa *interface.tk*, ver figura 4.3) é possível controlar os robots da comunidade remotamente e também:

1. alterar os valores dos controladores: kp, kd, ki, kl.
2. alterar a aceleração.
3. gravar e ler todos os dados anteriores.
4. alterar a sensibilidade do *space mouse* e do controlo (por ícones).

4.4.3 Módulo de jogo

Através do módulo de jogo é possível realizar vários jogos de futebol robótico com os elementos existentes na comunidade. Na interface (ver manual do utilizador), podemos escolher a posição de cada robot no campo (guarda-redes, atacante), podemos ainda definir qual a equipa (azul, amarelo) que pertence a cada robot. Com estas definições podemos realizar qualquer combinação de jogos, tais como, todos os robots contra uma equipa adversária (como em competição), três robots atacantes contra um guarda-redes. Esta funcionalidade é ideal para a realização de testes de campo. Na interface também é possível seleccionar as posições de defesa e meio-campo, embora estas posições ainda não existam, optou-se pela sua inclusão uma vez que a introdução destas posições de jogo está para breve.
Durante a execução do jogo é possível verificar o estado de qualquer variável existente no *blackboard* de qualquer robot, a este processo designa-se por telemetria, o qual iremos descrever na secção de telemetria.
Para efectuarmos este controlo sobre o jogo, quer a nível da inicialização, finalização de jogo e a nível de telemetria, foi necessário introduzir mais micro-agentes à arquitectura de *software* existente em cada robot (ver figura 4.1), nomeadamente:

Kontrol: este micro-agente é responsável pela inicialização do jogo, e também pela inicialização e finalização da tarefa de telemetria. A isto, junta-se a função de armazenamento das variáveis do *blackboard* que serão monitorizadas no processo de telemetria. Resumindo, o *Kontrol* é a entidade através da qual, a aplicação gráfica exerce remotamente controlo sobre o jogo.

***Telemetry*:** micro-agente responsável pelo processo de telemetria (ver figura 4.3). Após a selecção da posição e equipa de cada robot pretendido, é então possível dar-se início ao jogo. O utilizador pode comandar um jogo através da sequência dos seguintes comandos disponíveis na interface gráfica:

Ready-Game (modo de inicialização do jogo): comando *Ready-Game* lança uma tarefa de telemetria dentro da aplicação principal de modo a receber toda a informação de telemetria de toda a comunidade de robots, esta tarefa apenas é activada durante o modo de jogo. Seguidamente a aplicação envia uma mensagem ao processo *slave*-robot de cada robot seleccionado para jogar,

a mensagem contêm os parâmetros de jogo (informação de posição e campo) de cada robot. Após esta sequência a aplicação espera algum tempo (através da chamada à função *sleep*) de modo a garantir que o processo de jogo esteja em funcionamento. Seguidamente é necessário informar o processo de jogo de cada robot, através de mensagens para a tarefa *kontrol* (existente no processo jogo), de qual a informação (telemetria) que se pretende obter do *blackboard* de cada robot. Finalmente após toda esta sequência de operações o botão de *Start-Game* ficará disponível para o utilizador iniciar o jogo.

Start-Game (início do Jogo): o comando *Start-Game* é responsável pelo início do jogo, para tal a aplicação envia uma mensagem com esse fim para a tarefa *kontrol* de cada robot. A tarefa *kontrol* de cada robot ao receber esta mensagem vai: lançar a tarefa de telemetria se existir informação que se pretenda obter do robot em questão; libertar o semáforo da tarefa de visão responsável pelo início do jogo propriamente dito.

End-Game (finalização do jogo): a aplicação principal envia uma mensagem ao processo *slave*-robot para este terminar a execução do jogo. O processo *slave*-robot ao receber a mensagem do tipo fim de jogo, envia um *signal* ao processo de jogo para este terminar a execução. A aplicação também termina a tarefa de telemetria que coexiste no mesmo processo, esta tarefa apenas é lançada em modo de jogo.

4.4.3.1 Telemetria

A telemetria permite ao utilizador obter toda a informação sobre o estado actual dos robots quando estes se encontram a funcionar autonomamente.

Inicialmente já existia alguma telemetria suportada pela tarefa *monitor.c* . Anteriormente, o processo de inicialização do jogo era efectuado por um *script* que lançava a aplicação de jogo por rsh (*remote shell*), a qual era executada numa janela de *xterm*. Deste modo, o objectivo da tarefa *monitor.c* era debitar informação do robot ao utilizador através deste processo de rsh. Esta informação consistia numa sequência de linhas de caracteres composta pela concatenação de várias variáveis pré-definidas, em que cada linha reflectia uma iteração da máquina de estados. No entanto este tipo de telemetria baseava-se numa linguagem difícil de seguir. Foi então que decidimos tornar este processo de telemetria bastante mais amigável.

O utilizador pode escolher qual a informação (conjunto composto por inteiros ou cadeias de caracter) que deseja obter do *blackboard* e a que taxa de actualização esses que mesmos dados são transmitidos. Esta funcionalidade permite ter dados diferentes a serem actualizados a velocidades de refrescamento diferentes. O facto dos modems sem fios utilizados na comunicação apresentarem uma pequena largura de banda (cerca de 2Megabits/sec), conjuntamente com o facto dos robots usarem os modems fundamentalmente para os aspectos relacionados com o jogo, motivou a criação de um dispositivo de telemetria que, primeiro, libertasse os modems em termos de largura de banda para o que é realmente importante, o jogo, e segundo, efectuar a telemetria de dados que são considerados mais importantes e/ou dinâmicos

em detrimento dos dados que são menos importantes e/ou estáticos. No entanto, é ao utilizador que cabe a total liberdade de escolha de qualquer informação que deseje obter valores, de qualquer robot em jogo, e atribuir a esses mesmos dados, taxas de actualização que reflictam a importância dos mesmos. A taxa de refrescamento corresponde a um inteiro em unidades de 1/10 *sec*. Consideramos que esta unidade é bastante satisfatória como base para a transmissão de informação da telemetria.

Na figura 4.7 esquematiza de uma maneira global como o processo de telemetria funciona.

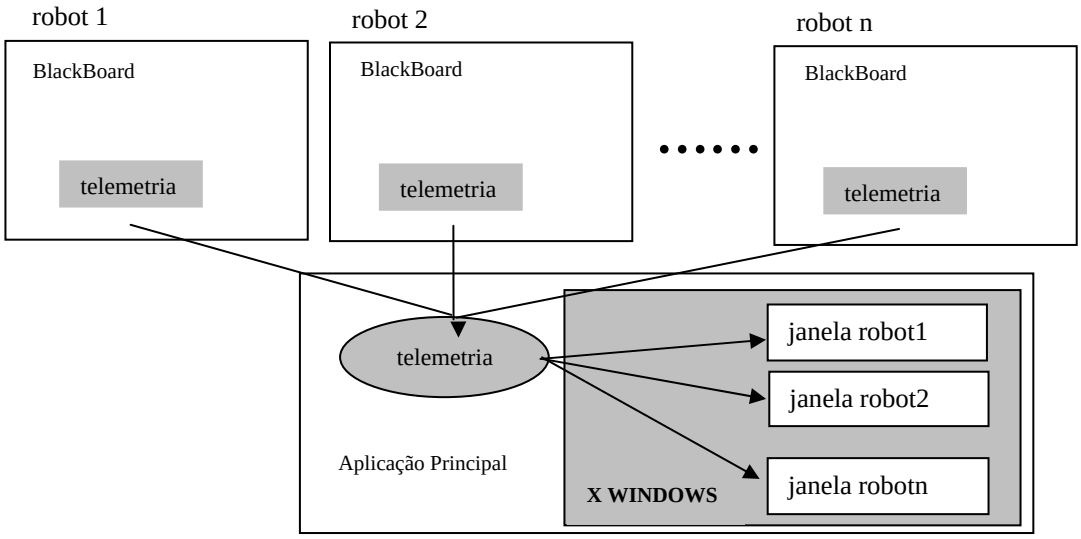


Figura 4.7: funcionamento do processo de telemetria

Do lado do robot existe uma tarefa que consiste num ciclo que efectua a leitura periódica (definida pelo utilizador) da informação pretendida do *blackboard*. Posteriormente envia esses dados para a tarefa de telemetria da aplicação principal através da rede. Do outro lado, na aplicação principal, a tarefa de telemetria é responsável pelo reencaminhar da informação transmitida por cada robot para a janela X correspondente a esse robot. Apesar de toda a interface gráfica ter sido implementada em Tcl/Tk, optou-se por usar directamente a biblioteca gráfica *X-Windows* como meio para mostrar a informação proveniente do *blackboard* de cada um dos robots, uma vez que esta biblioteca contêm as primitivas gráficas de mais baixo nível dos sistemas baseados em *Linux* e consequentemente uma maior velocidade de actualização dos dados em modo gráfico.

4.4.4 Módulo de Transmissão da imagem dos robots

O módulo de transmissão da imagem é aqui tratada com sendo uma aplicação independente de tudo o resto, o que de certa forma até corresponde à realidade, uma vez que este (e outros módulos) foram implementados de forma a manterem a independência com o resto da aplicação, sendo apenas integrados na interface gráfica. Assim, este módulo pode ser facilmente compilado e executado isoladamente, mas

também ser facilmente integrados noutras aplicações futuras a serem desenvolvidas neste ou noutro projecto.

4.4.4.1 Introdução

O mecanismo de transmissão da imagem que é captada pela câmara de um robot e o servidor central, divide-se em dois processos distribuídos. Cada um desses processos contém por sua vez duas tarefas que operam entre si da forma mais concorrente possível, mas sempre em sincronismo. Do lado do robot, uma tarefa capta a informação da câmara e escreve-a num espaço de memória, enquanto outra se encarrega de a ler e enviar através da rede para o servidor. Do lado do servidor, uma tarefa encarrega-se de receber os pacotes vindos da rede e a escreve-los num espaço de memória, enquanto que outra mostra a informação (em forma de imagem) no ecrã.

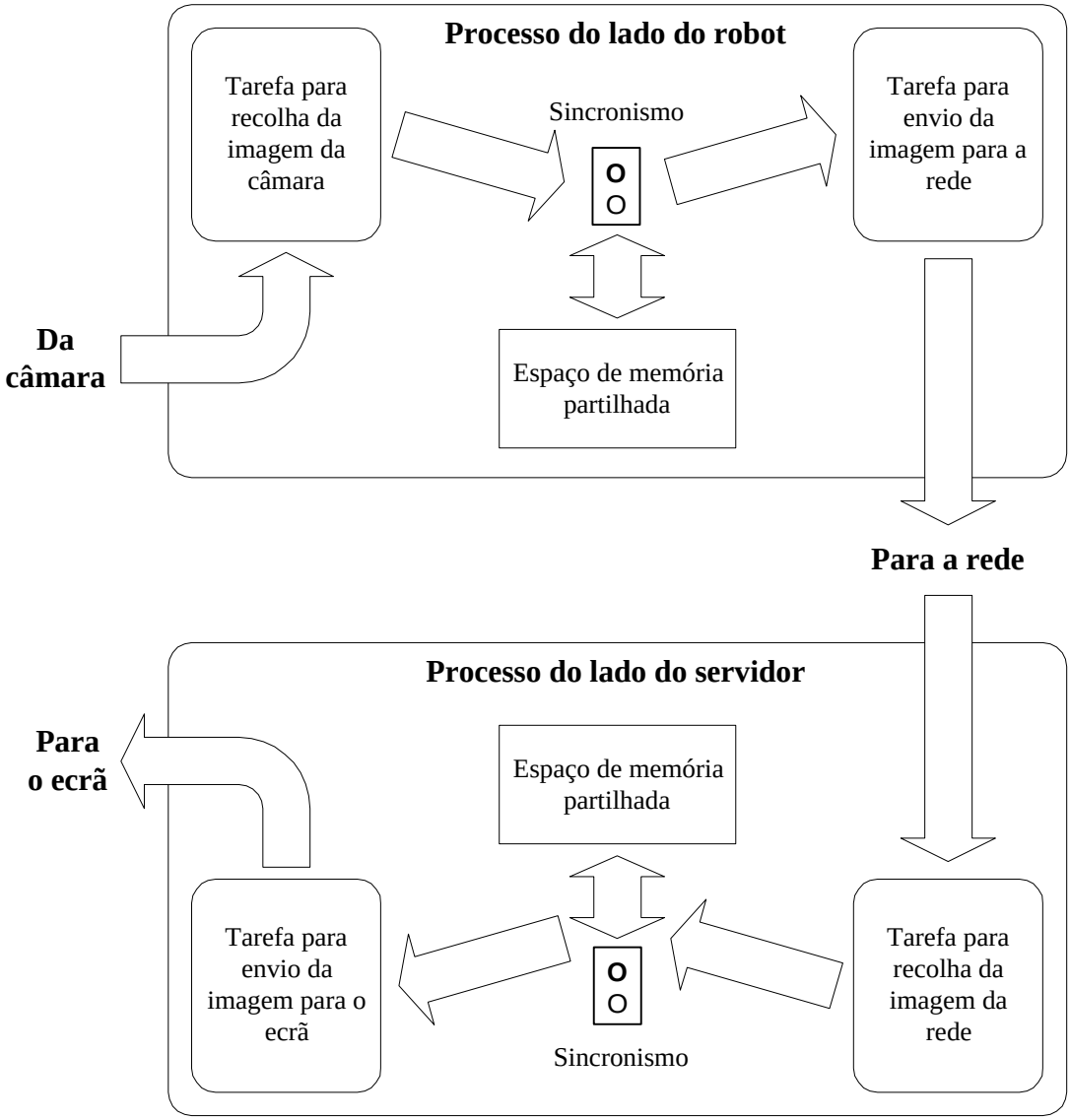


Figura 4.8: Arquitectura global do sistema de captura da imagem pela câmara de vídeo, seu envio e recepção através da rede, e correspondente apresentação no ecrã

A parte da aplicação que capta a imagem da câmara de video para a memória já existia no projecto (no ficheiro *hal.c*), nomeadamente existe um micro agente (*vision*) que se encarrega de captar e processar as sucessivas imagens captadas. As funções existentes são portanto utilizadas também neste contexto.

4.4.4.2 Paralelismo: processos ou tarefas?

Ao nível do sistema operativo, aquilo a que na figura 4.8 se chama de processo, pode corresponder a uma das seguintes implementações:

1. A um conjunto de dois processos que recorrem a um mecanismo de memória partilhada para troca da informação sobre as imagens.

2. A um único processo único que cria duas tarefas (*threads*), que partilham o mesmo espaço de endereçamento.

☞ Para consultar um exemplo da criação de dois processos recorrendo à primitiva *fork* ver anexo A, secção A1. Para o caso das tarefas, consultar o exemplo do apêndice A, secção A2.

☞ Para consultar um exemplo de utilização de memória partilhada entre dois processos *Linux* recorrendo a semáforos para sincronismo no acesso, ver apêndice A, secção A3.

Um factor de simplicidade de implementação entre processos *versus* tarefas, tem a ver com a troca de mensagens de controle entre as tarefas em oposição aos processos. No caso das tarefas, quando a tarefa que mostra a imagem no ecrã quiser iniciar ou suspender a recepção de imagens tem de comunicar à tarefa que as recebe da rede. Neste caso, essas mensagens podem ser trocadas recorrendo a variáveis de estado globais da aplicação que de tempos a tempos são analisadas pelos respectivos intervenientes. Para o caso dos processos independentes, terá de se usar o espaço de memória partilhada para trocar as tais variáveis de estado, ou então estabelecer um qualquer protocolo associado a um mecanismo de comunicação entre ambos os processos intervenientes.

Uma solução eficiente para implementar esse mecanismo de comunicação, e que foi implementada na primeira solução, será recorrer à troca de *signals Linux* entre os processos. Nomeadamente os *signals* reservados para utilização por parte dos programadores (SIGUSR1 e SIGUSR2). Esta solução tem a vantagem de que, sendo os *signals* eventos assíncronos, quando se desejar enviar uma mensagem, os efeitos serão imediatos. No caso de se implementar um mecanismo usando variáveis globais entre as tarefas, ou variáveis residentes no espaço de memória partilhada pelos processos, o efeito do envio da mensagem só ocorrerá quando o processo que as lê for analisar esse conjunto de variáveis. Esta foi a primeira solução avançada.

☞ Para consultar um exemplo de troca de *signals* entre dois processos *Linux* recorrendo à primitiva *kill* ver apêndice A, secção A4.

Após a aplicação estar implementada e robusta na sua execução, foi repensada a estratégia utilizada: dois processos que recorrem a memória partilhada com envio de *signals* para sincronismo, decidiu-se evoluir para uma solução orientada às tarefas, com um processo único.

O reajustamento do código de uma solução para outra foi bastante trivial e rápida. De seguida enumeram-se os passos seguidos para o efeito:

1. Juntaram-se os ficheiros que correspondiam aos dois processos existentes num único ficheiro, e procedeu-se a pequenos ajustamentos como sendo remover os ficheiros incluídos duplicados, etc.

2. Removeu-se o código correspondente à inicialização e gestão da memória partilhada.

3. Criou-se uma estrutura de dados que contém a informação a ser partilhada pelas tarefas (que substitui a memória partilhada), como sendo os apontadores para as zonas de memória utilizadas, lista de agentes ligados ao sistema, referência ao agente activo, etc.

4. Alterou-se o nome das duas funções principais (*main*) existentes de forma a transformá-las em funções secundárias.

5. Recorreram-se a pequenas adaptações por forma às funções passarem a operar com a estrutura criada e não com a memória partilhada (foi apenas necessário adaptar os nomes dos apontadores que anteriormente apontavam para uma zona de memória partilhada, e agora apontam para os elementos da nova estrutura). A sincronização não foi alterada.

6. Criou-se uma nova função principal, onde se executa o *broadcast* inicial, e que de seguida cria e passa o processamento às duas tarefas que correspondiam aos antigos processos.

4.4.4.3 Aumento do desempenho do sistema

Para se optimizar o número de imagens enviadas para a rede, e mostradas por segundo no ecrã, é utilizada uma técnica aqui chamada de *double buffer*. Ou seja, cada uma das zonas de memória partilhada contém dois segmentos que são escritos/lidos alternadamente pelas tarefas de forma a que quando uma tarefa está a escrever num segmento, a outra pode estar a ler do outro. Esse acesso terá de ser sempre feito de forma sincronizada.

Se a leitura da informação que vem da rede através dos *modems* utilizados para a memória, e a leitura da memória para o ecrã, for feita recorrendo ao mecanismo DMA (*direct memory access*), utilizando canais diferentes, ou seja, sem que a informação passe pelo processador, então será possível aumentar bastante o desempenho da aplicação uma vez que essas leituras/escritas serão feitas realmente

em paralelo, e não em *time-sharing*, ou outra qualquer forma de partilha do processador. Tendo em conta que as escrita ou leituras de ou para os *modems* será sempre (muito) mais lenta que a leitura da câmara para a memória ou da memória para o ecrã, então no limite, o desempenho da aplicação tenderá para uma função que terá como única variável livre, o desempenho dos *modems*.

Este meio de paralelismo apresenta no entanto uma desvantagem: uma imagem aparece no ecrã ligeiramente atrasada no tempo em relação ao momento em que foi captada, uma vez que a imagem que é apresentada não é a ultima que foi recebida: a última a ser recebida estará nesse dado instante a ser lida da rede pela outra tarefa. Apesar disso, o paralelismo aumenta o número de imagens por segundo que são apresentadas.

Para consultar um exemplo de utilização do *double buffer* entre dois processos *Linux*, recorrendo a sincronismo, ver apêndice A, secção A3.

4.4.4.4 Sincronismo

O mecanismo de sincronização baseia-se num algoritmo bastante conhecido na literatura como o de "produtor - consumidor", em que as tarefas de escrita da informação vinda da câmara ou da rede para a memória são as tarefas produtoras; as tarefas de leitura da informação para as enviarem para a rede ou para o ecrã as tarefas consumidoras. O único detalhe que diferencia este algoritmo do já conhecido, será a utilização em alternância de duas zonas de memória.

Informalmente, este mecanismo pode ser explicado através de um exemplo muito simples (figura 4.9): as duas tarefas alternam de segmentos de memória. Quando por exemplo, a tarefa de escrita A acaba de produzir no segmento de memória respectivo, destranca (liberta o semáforo) o mesmo segmento e alterna para o segmento seguinte. Se esse outro segmento ainda estiver trancado pela tarefa B então bloqueia-se até que a mesma tarefa de leitura B que o está a consumir o liberte. Quando isso acontecer a tarefa de leitura B passa a consumir sobre o antigo segmento de memória da tarefa de escrita A (já destrancado anteriormente), e a tarefa de escrita A desbloqueia-se e passa a produzir sobre o segmento de memória que a tarefa de leitura B acabou de libertar. Este mecanismo é descrito em pseudocódigo por baixo. Note-se que o mesmo algoritmo apresentado é muito simplista e informal, mas ilustra a ideia base do mesmo. As primitivas apresentadas são:

1. Espera: tenta aceder a uma zona de memória. Caso esta esteja trancada, bloqueia o processo até ser destrancada. De seguida tranca a zona de memória e fica livre para lhe aceder

2. Produz/Consome: executa as operações sobre a memória.

3. Assinala: Liberta as zonas de memória trancadas anteriormente permitindo a que estas possam ser usadas por outros.

Tarefa de escrita A:

```
Zona de memória = 1

Ciclo infinito

      Espera memória N
      Produz memória N
      Assinala memória N

      Troca de memória

Fim ciclo.
```

Tarefa de escrita B:

```
Zona de memória = 2

Ciclo infinito

      Espera memória N
      Consome memória N
      Assinala memória N

      Troca de memória

Fim ciclo.
```

Figura 4.9: exemplo (simplista) da implementação de um mecanismo de produtor - consumidor usando duas zonas de memória.

Para consultar um exemplo de utilização de um *double buffer* de memória partilhada entre dois processos *Linux*, recorrendo a sincronismo, ver apêndice A, secção A3.

4.4.4.5 Envio da imagem pela rede

Para a comunicação entre os robots e eventualmente um servidor, está disponível uma rede *ethernet* sem fios.

Para o envio dos pacotes de imagem, optou-se por usar o protocolo *sockets* sem ligação.

A razão para se usar *sockets* sem ligação (*datagram*), e não com ligação (*stream*), tem a ver fundamentalmente com eficiência devido ao não controle de erros (percas, sequência, duplicados, etc.), muitas vezes à custa de pedidos de retransmissão, associado aos *sockets* com ligação.

Uma vez que sendo o envio de imagens tipicamente bastante pesado no que diz respeito à quantidade de informação transmitida, todo o tempo que se poupar em controle (por vezes inútil) pode ser importante para um desempenho aceitável do sistema.

Esse controle é na realidade inútil a maioria das vezes. Imagine-se por exemplo que se perdia uma imagem na rede. O protocolo *stream* passado um temporizador, iria pedir a retransmissão da imagem perdida. Pior ainda, se um dos *bytes* da imagem viesse corrompido, o protocolo iria também pedir a retransmissão da imagem (caso não o corrija automaticamente com base em informação redundante). No entanto, não é este o comportamento que se espera de uma transmissão de imagem em tempo real. Ou seja, se um *pixel* da imagem vier corrompido, não se está à espera que a recepção de imagens pare para se pedir uma retransmissão. O mesmo acontece se uma imagem inteira se perder na rede: a recepção das próximas imagens terá de

continuar! Sendo assim, optou-se por utilizar os *sockets* sem ligação, assumindo-se algumas propriedades do meio como válidas à partida:

1. Não se geram duplicados. Ou seja, um pacote é recebido uma e uma só vez ou não é recebido, não acontecendo a situação de um mesmo pacote ser recebido duas ou mais vezes (o que é perfeitamente aceitável dadas as características dos *modems* utilizados).

2. A sequência de recepção é igual à sequência de envio. Ou seja, não existe perca de sequêncialidade. Este factor poderá ser bastante importante uma vez que (como se verá mais adiante) sendo a imagem partida em faixas, há que garantir que uma faixa não chega primeiro que outra, dando origem a uma imagem estragada (mesmo que isto aconteça uma ou outra vez pode não ser muito grave, se for apenas uma imagem que chega estragada, entre muitas outras que chegam em boas condições). No entanto, este problema é controlado recorrendo a um algoritmo da aplicação (como se verá mais adiante).

O maior problema da implementação do mecanismo de envio da imagem através da rede entre máquinas teve a ver com as limitações de ordem física dos *modems* utilizados. Observou-se na prática que quando se tentam enviar grandes quantidades de informação (como sendo uma imagem mesmo de pequena resolução) de uma forma sistemática, os *modems* começam a descartar os pacotes enviados. Ou seja, o problema aparenta ter duas vertentes: primeiro não é possível enviar pacotes de grandes dimensões, nomeadamente os modems não suportam o tamanho máximo de pacote definido nas bibliotecas de *sockets datagram* do *Linux*. Para além disso, os *modems* não suportam que se lhe esteja a enviar informação de dimensão razoável, de forma sistemática (ao fim de um certo tempo começam a descartar pacotes). Este facto acontece possivelmente devido aos *modems* aparentemente disporem de uma capacidade bastante limitada de *buffers* internos, pelo que quando se tentam enviar grandes quantidades de informação estes ficam rapidamente lotados antes de os despacharem para a rede, ou no caso limite nem suportam um pacote de grandes dimensões, e logo começam a descartar os pacotes aos quais não tem capacidade para dar resposta.

De qualquer forma, e independentemente do causa exacta que leva ao comportamento pouco amigável dos *modems* em questão, foi implementada uma solução para o problema. A solução consiste primeiro em partir a imagem em faixas para se passarem a enviar pacotes de dados mais pequenos. De seguida não estar sistematicamente a enviar pacotes de informação: esperar por uma confirmação do lado oposto para continuar o mecanismo de envio. Concretamente a imagem é partida em tantos bocados quantos os especificados no ficheiro de configuração da aplicação. De seguida é enviada uma faixa da imagem para a rede, e o processo bloqueia-se até receber resposta do receptor para continuar o envio. O servidor recebe essa imagem, preenche a parcela correspondente do segmento de memória, e envia a confirmação da recepção a fim que o processo do lado do robot continue a enviar faixas de imagem.

Para consultar um exemplo de utilização de *sockets* para comunicação entre dois processos *Linux*, ver apêndice A, secção A5.

Para este propósito, e também para controlo das ligações (inicio, pausa do envio da imagem, etc.) foi definido um protocolo bastante simples de mensagens. Esse protocolo possui um conjunto de mensagens de dados:

IMG_OLA: mensagem enviada em difusão do servidor, para todos os robots disponíveis. Tem a finalidade de informar os robots que está disponível um servidor para estes comunicarem.

IMG_PAR: mensagem enviada de um robot para o servidor. Tem a finalidade de informar o servidor que o robot que a enviou está activo e também envia para este os parâmetros relativos ao envio/recepção da imagem do robot para o servidor. Esses parâmetros são: as dimensões em termos de (xmax, ymax) de uma imagem, o número de faixas em que a imagem é partida, o número de *bytes* por pixel. Quando o servidor recebe uma mensagem deste tipo, insere o robot na sua lista de agentes disponíveis. Assim, é possível haver configurações diferentes para cada robot.

IMG_INIC: mensagem enviada do servidor para o robot. Tem a finalidade de pedir ao robot para iniciar o envio de imagens para o servidor.

IMG_IMG: mensagem enviada do robot para o servidor. Corresponde a uma mensagem de dados que contém uma faixa de imagem.

IMG_CONF: mensagem enviada do servidor para o robot. Tem a finalidade de informar o robot que a ultima faixa de imagem chegou em boas condições, e pede o envio da próxima.

IMG_SUSP: mensagem enviada do servidor para o robot. Tem a finalidade de pedir ao robot para suspender o envio de imagens para o servidor.

Desta forma, pode-se desenhar a máquina de estados (bastante simplista e informal) da aplicação global.

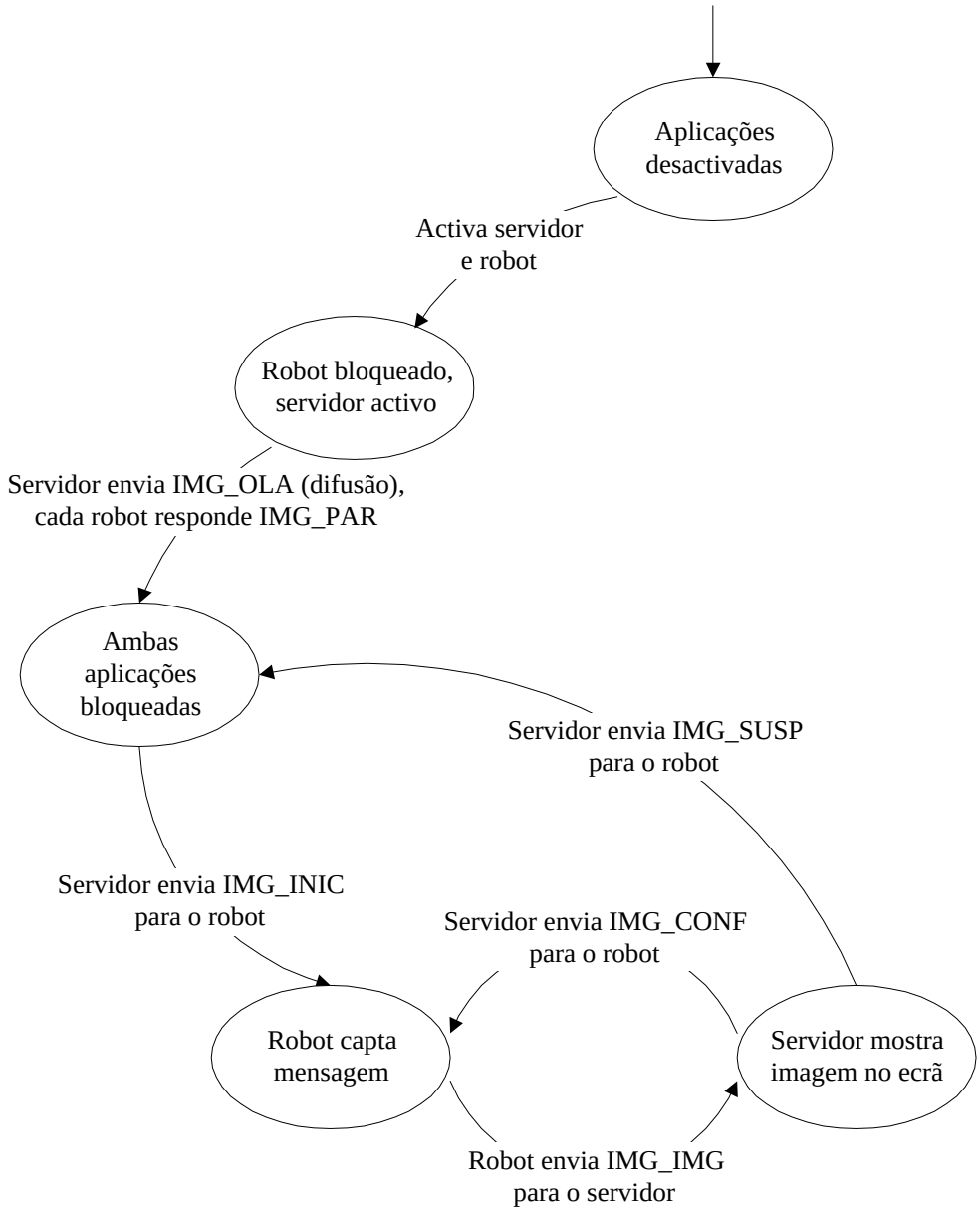


Figura 4.10: máquina de estados (simplista) do protocolo do envio/recepção de imagens pela rede.

☞ Para consultar um exemplo de utilização destas funções de conversão de dados, ver apêndice A, secção A6.

4.4.4.6 Replicação do lado dos robots

Para ser possível captar a imagem num servidor remoto, é necessária a presença do lado dos robots de uma aplicação que capta e envia a imagem, como já foi visto. Essa aplicação deverá estar preparada para, em caso de falha, não morrer

definitivamente. Ou seja, em caso de falha grave, no limite a aplicação deverá ter a capacidade de se reinicializar. Este propriedade pode ser implementada interceptando os *signals* que a aplicação recebe em caso de acontecimentos de excepção que levam à terminação do programa, executando de seguida os mecanismos necessários para restabelecer o bom funcionamento do sistema. Contudo este método é muito pouco robusto, e não dá garantias de fiabilidade por duas razões fundamentais:

1. Nem todos os *signals* podem ser interceptados: SIGKILL e SIGSTOP.

2. O comportamento de um processo é indefinido após este ignorar um SIGFPE, SIGILL ou SIGSEGV que não foi gerado pelas primitivas *kill* ou *raise*.

O ideal seria ter um mecanismo que se encarregaria de supervisionar o comportamento do processo que está a operar, e que em caso de falha, se encarregar de o restabelecer.

Foram detectadas duas soluções:

1. Inserir a aplicação no processo *inetd* (*internet super-server*). Essencialmente o *inetd* corre um *daemon* que invoca uma série de outros, reduzindo a carga do sistema. Através do *inetd*, é possível associar uma aplicação a um *socket*. Essa atribuição *socket* contém o nome da aplicação que deve ser invocada, quando um pedido é encontrado nesse mesmo *socket*. Para o efeito é necessário configurar um ficheiro (*/etc/inetd.conf*) com os parâmetros para a aplicação que se quer inserir. Desta forma, caso a aplicação termine de forma inesperada, quando surgir um novo pedido a essa aplicação via *sockets*, o *inetd* encarregar-se-á de recarregar a aplicação.

2. Implementar um mecanismo de replicação activa. Ou seja, criar duas aplicações, uma primária A e uma secundária B. A aplicação secundária B é a aplicação já conhecida que capta e envia as imagens. A aplicação primária A, será uma aplicação que monitorizará a aplicação secundária B (através de troca de mensagens de provas de vida), e que quando detecta que a aplicação secundária B terminou de uma forma inesperada, lança outra aplicação secundária. Este mecanismo, parecendo complicado de implementar (nomeadamente o protocolo de troca de mensagens de prova de vida), é bastante trivial uma vez que o sistema operativo dispõem de bibliotecas com primitivas que permitem implementar a solução de uma forma quase automática.

A escolha recaíu na segunda solução.

Para consultar um exemplo de utilização de desta última solução de replicação, ver apêndice A, secção A7.

5. EVOLUÇÃO DA COMUNIDADE DE ROBOTS FUTEBOLISTAS

Neste capítulo é efectuada a descrição da evolução dos robots *SocRob*, ao nível da arquitectura física, durante a nossa permanência na equipa *ISocRob*. Também é descrita a forma como foi feita, por nós, a integração de um robot comercial *Super Scout II* na equipa.

5.1. Enquadramento e evolução dos robots *SocRob*

O trabalho integra-se, como já foi dito no projecto *ISocRob* (*Soccer Robots* ou *Society of Robots*). A equipa havia já participado no ano passado no evento anual de futebol robótico realizado na altura em Paris. Já existia então uma plataforma de *hardware* concretizada em três robots todos eles semelhantes entre si: a plataforma *SocRob*. Esta plataforma apresenta as seguintes características (resumidas):

Dimensões físicas: 410mm x 410mm x 360mm.
Placa mãe: AMDK6 200MHz.
Placa de raiz para controlo dos motores, baseada do controlador de motor LM 629
Placa de rede *Realtek Ethernet* 10Mb/s
Modems Ethernet: RF sem fios da *AAEON*
Câmara de vídeo: *Captivator* PCI/VC.
Cinemática: controlo diferencial.

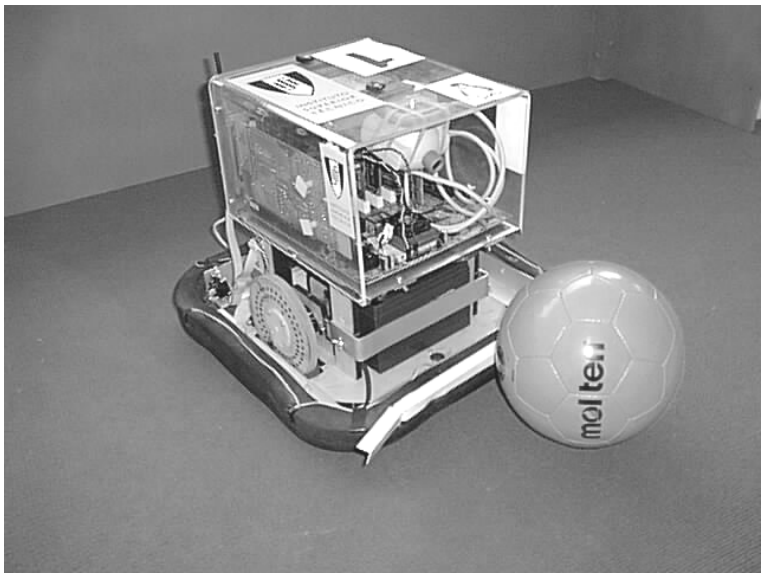


Figura 5.1: plataforma *SocRob* (1) modelo mais antigo.

Os robots de arquitectura *socrob* foram, no entanto, sujeitos a melhorias por parte de outros elementos da equipa, destacam-se as de maior relevância, nomeadamente:

1. colocação de rodas mais leves e com *o-rings* que permitiam uma maior aderência ao chão.

2. dispositivo de chuto da bola, este dispositivo é controlado por um sistema *PIC*, em que a comunicação com o computador é efectuada através da porta série.

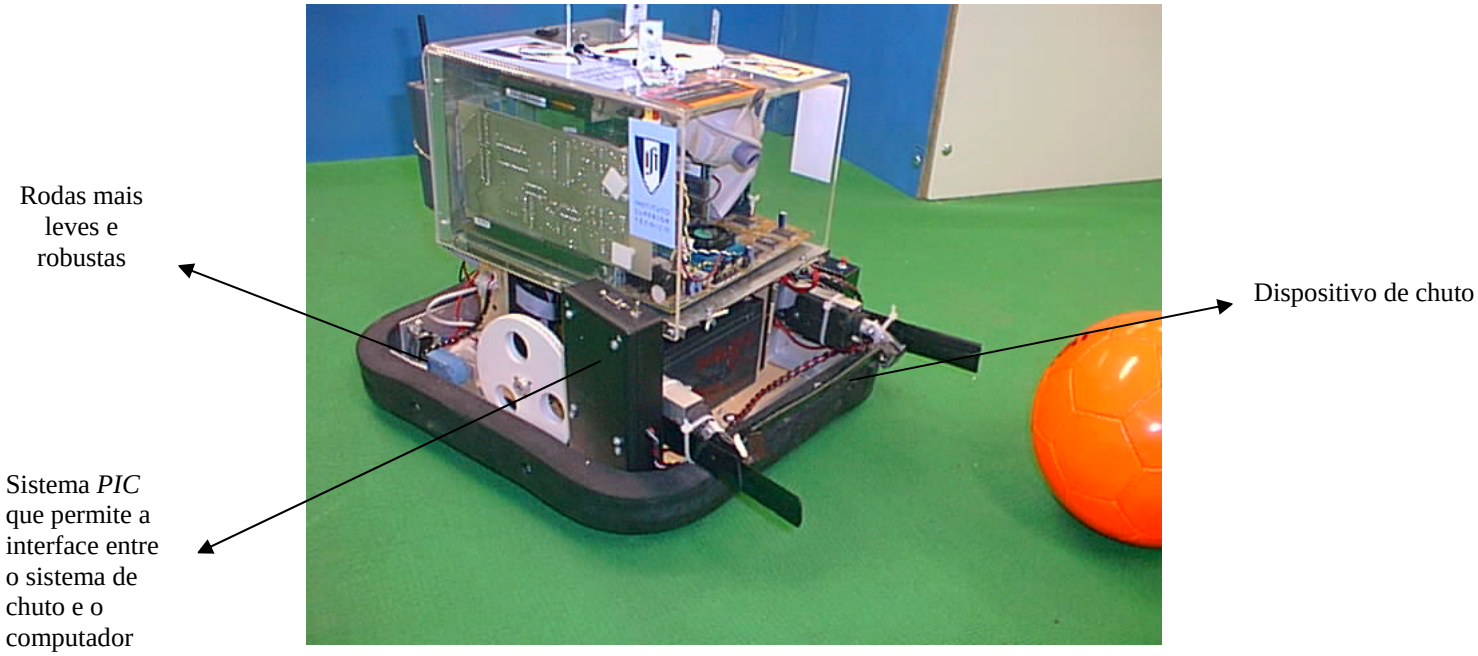


Figura 5.2: plataforma *SocRob* (2) modelo mais recente

5.2. Integração de um robot comercial *Super Scout II* na comunidade

Á comunidade de robots existente foi acrescentado um elemento: um robot *Super Scout II* [19] [20] (ver apêndice B). Esse elemento difere dos restantes na medida em que ao contrário dos outros é uma plataforma comercial já com grande parte do *hardware* funcional, não sendo portanto necessário a construção de uma plataforma de raiz. No entanto foram necessárias algumas adaptações no *hardware* e transportar o *software* já existente nos outros robots para este novo elemento.

5.2.1. Introdução

Tendo em vista a já referida participação da equipa *ISocRob*, onde este projecto se integra, no evento anual de futebol robótico, foi necessário acrescentar um

elemento à comunidade de robots existente. Esse aumento , deveu-se à alteração do número de robots por equipa que este ano passaram a quadro. Até ao momento, a equipa dispunha de três robots construídos de raiz: as plataformas *SocRob*. Para aumentar o desempenho da equipa, melhorando a competitividade e fiabilidade, mas também para poupar uma quantidade significativa de esforço na construção de uma nova plataforma, os lideres da equipa decidiram-se por uma plataforma comercial *Nomad Super Scout II*. Esta decisão recaiu sobretudo pela robustez da plataforma a mover-se em ambientes hostis, como sendo embates violentos, etc. Também é uma plataforma bastante fácil de manusear e com um movimento bastante suave, para além de ser já relativamente conhecida por alguns elementos ligados ao ISR. Assim, foi necessário transportar as aplicações existentes para esta nova plataforma por forma ao robot apresentar um comportamento coerente com o resto da equipa.

Quando o robot ficou disponível (duas semanas antes da competição) o trabalho começou por uma fase de familiarização da plataforma, nomeadamente as componentes de *hardware* e de *software* existentes. Foi também analisada de forma exaustiva a documentação da *Nomadic Technologies* [20] que acompanhava o robot. De seguida foram efectuados uma série de testes como por exemplo, testes ao controlo do movimento (acelerações, controladores, velocidades), e testes à imagem já que foi necessário obviamente instalar um câmara, tal como já existia nos outros robots da comunidade.

Começamos por efectuar uma descrição das especificação técnicas do Robot *Nomad Super Scout II* na secção A2. Na secção A3 enumeramos as alterações físicas que a plataforma foi sujeita de modo a adaptar-se ao futebol robótico. segue-se na secção A4, uma breve descrição da adaptação do software existente para o robot *Nomad Super Scout II* . Finalmente, na secção A5 concluiremos sobre a utilização deste tipo de robots a quando da sua utilização para o futebol robótico.

5.2.3. Alterações físicas à plataforma

De modo a tornar o robot *Super Scout II* um robot futebolista funcionalmente idêntico aos restantes da comunidade, foi necessário proceder a alguns melhoramentos à estrutura inicial, nomeadamente:

1. Foi retirado um sonar (este tipo de sensores por opção não são usados pela equipa) da parte frontal da estrutura e, no seu lugar foi introduzida um sistema de visão: uma câmara idêntica à dos outros robots.

2. Foram acrescentados uns suportes para transportar a bola, também gentilmente designados de bigodes por parte de alguns elementos da equipa.

3. Foi colocado um apoio para o modem sem fios

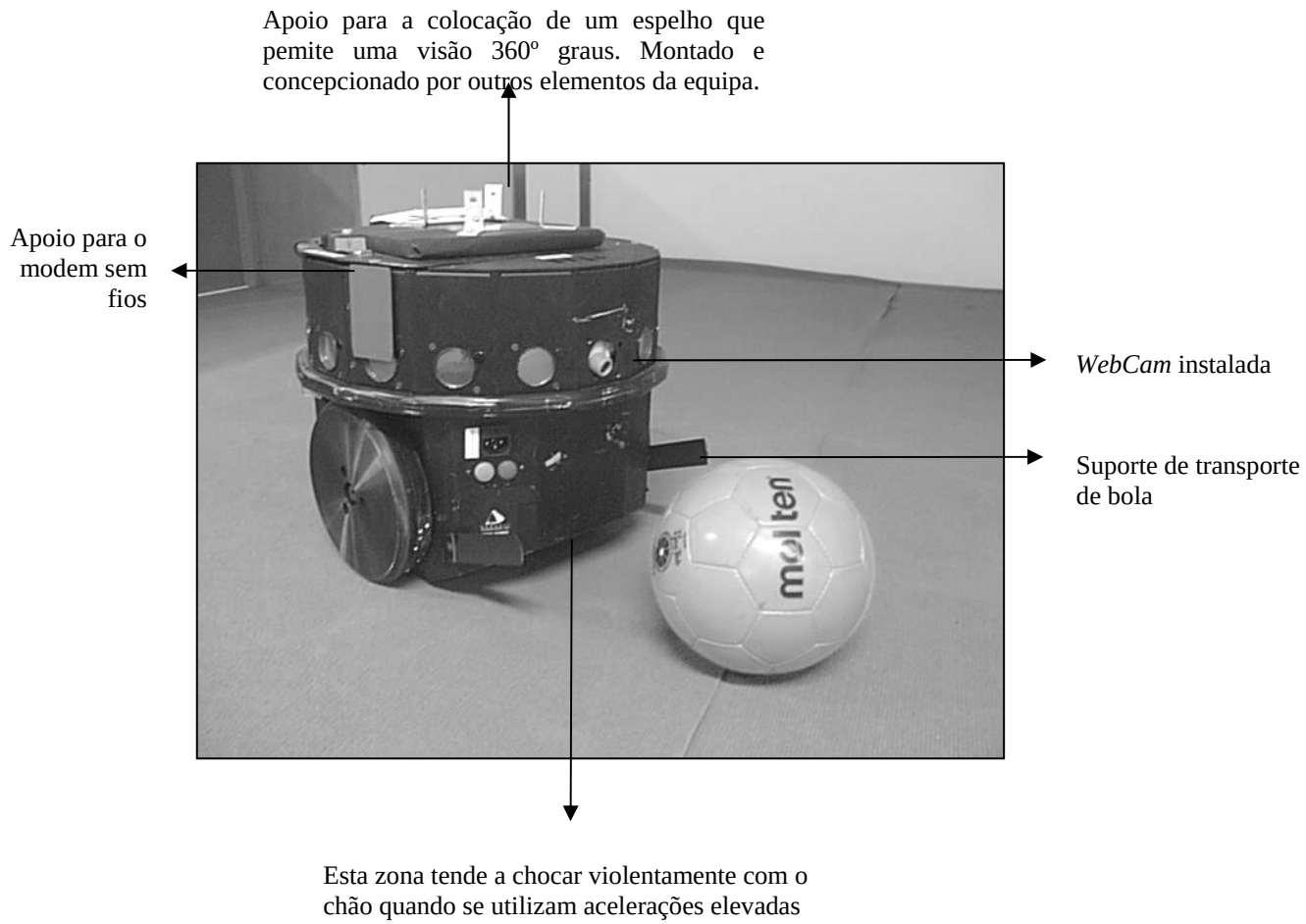


Figura 5.3: alterações físicas ao robot comercial *Super Scout II*

5.2.4. Possíveis melhoramentos à estrutura:

Nos testes que realizámos com acelerações elevadas constatámos alguma instabilidade na estrutura, nomeadamente na parte frontal inferior, para compensar esta instabilidade pensamos que deveriam ser colocadas nessa zona duas rodas de apoio, uma em cada extremo, ou apenas uma roda central. Para uma possível instalação de um sistema de chuto neste tipo de plataforma, têm-se que considerar previamente a colocação das rodas atrás referidas com o intuito de suportar o peso que daí possa advir.

5.2.5. Adaptação do *software* existente à plataforma *Super Scout II*

O *hal* (*hardware abstraction layer*) existente apenas fornecia uma camada de abstracção à plataforma *SocRob*, a única disponível. Optou-se por tornar este *hal* genérico de modo a suportar qualquer tipo de plataforma de forma escalável e robusta. Então, numa fase de inicialização dos motores, todas as funções de chamadas aos mesmos foram convertidas em ponteiros para funções relativas aos motores da plataforma pretendida. O mecanismo de redirecionamento dos ponteiros foi efectuado através de uma função de auto-detecção de plataforma. Em baixo na figura 5.4 e 5.5 é ilustrado como exemplo, o mecanismo de inicialização da função *motSetVel*() numa plataforma *Super Scout II*.

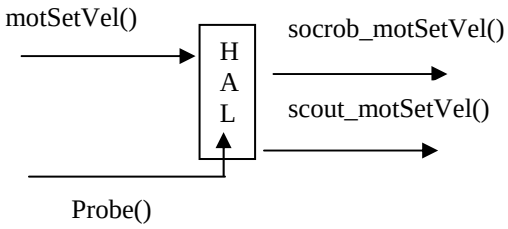


Figura 5.4:
A função de auto-detecção *Probe*() redirecciona o ponteiro *motSetVel* para a função referente à plataforma detectada

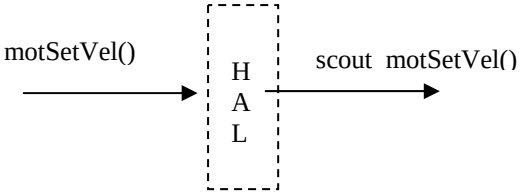


Figura 5.5:
A função *Probe*() detectou uma plataforma do tipo *Super Scout II*

Todas as funções relativas à plataforma *Super Scout II* foram programadas usando o *software* "*Linux-based*" de desenvolvimento de *software* da *Nomadic Techonogies* , o que facilitou bastante a tarefa de integração.

Dado que todo o código existente foi elaborado tendo em vista a utilização em plataformas *socrob*, todas as chamadas a funções de motores utilizavam referências tendo em vista esta plataforma. Quanto foi introduzido o robot *Super Scout II* na equipa, e para manter a coerência e o significado do *hal*, todas as funções que eram aplicadas à plataforma *Super Scout II* foram submetidas a um factor de escala de velocidade em relação à plataforma *socrob*. Para encontrarmos este factor de escala fizemos percorrer uma distância de 5 metros os dois robots (um *Super Scout II* e um *SocRob*) a uma dada velocidade constante, medidos os tempos, bastava aplicar uma regra de três simples para obtermos o factor de escala pretendido, sublinhe-se que este resultado apenas foi possível devido à linearidade dos sistemas. Na figura 5.6 está esquematizado a obtenção dos valores que permitiram obter o factor de escala. Dado que só se utiliza uma única aceleração para qualquer comportamento de jogo, optou-se por determinar este valor de uma forma empírica para a plataforma *Super Scout II*.

Plataforma	Velocidade de comando	distância ≈	tempo	velocidade real
SocRob	100000	2m	9 sec	22cm/s
Super Scout II	100	2m	8 sec	25cm/s

Figura 5.6: quadro dos factores de escala entre as plataformas *SocRob* e *Super Scout II*

Logo, para uma plataforma *Super Scout II*,uma velocidade real de 22cm/s corresponde a uma velocidade de comando de 88, portanto o factor de escala que permite mapear velocidades da plataforma *SocRob* para a plataforma *Super Scout II* é dada por:

$$\text{factor de escala} = \frac{88}{100000}$$

A velocidade de comando para a plataforma *Super Scout II* é dada então por:

$$\text{Vcom (}\textit{Super Scout II}\text{)} = \frac{88}{100000} * \text{Vcom (}\textit{socrob}\text{)}$$

5.2.6. Resultados e Conclusões

Durante a fase de testes à plataforma *Super Scout II* constatámos que as velocidades e acelerações mencionadas no manual *Language Reference Manual* da *Nomadic Technologies inc.* eram meramente teóricas (que devem corresponder a um limite de protecção), podendo a plataforma suportar velocidades e acelerações muito superiores. Assim sendo é possível termos acelerações até 50 polegadas/sec2 enquanto o manual refere apenas 30 polegadas/sec2, e termos velocidades de 45 polegadas/sec enquanto no manual refere 24 polegadas/sec. Os valores apresentados foram obtidos no limiar da saturação, não devendo no entanto ser aplicadas referências superiores de modo a não colocar em causa a integridade do robot.

Dada a brilhante e não menos magnífica participação do robot *Super Scout II* no campeonato mundial de futebol robótico realizado este ano em Estocolmo, podemos dizer de nossa justiça e pela experiência neste tipo de plataforma comercial que, o *Super Scout II* é um excelente robot para a utilização no futebol robótico. A robustez e fiabilidade da estrutura são sem margens para dúvida os seus pontos fortes, que nos libertam dos problemas de hardware e nos focalizam no que é importante nesta matéria: o *software*, nomeadamente de toda a inteligência artificial que está por detrás deste tipo de actividade. O único senão é a velocidade. O *Super Scout II* não é muito veloz em relação aos seus competidores na cena actual, por isso consideramos que este tipo de plataforma deverá ser utilizada em qualquer posição no campo à excepção da posição de guarda-redes. No campeonato, o *Super Scout II* superou todas as expectativas ao marcar todos os golos à excepção de um em que assistiu.

6. TRABALHOS FUTUROS

O trabalho integra-se num projecto de longo prazo que espera-se terá continuidade por ainda alguns anos. Sendo assim, é natural que emirjam ideias, sugestões e comentários críticos por parte das pessoas que nele participaram com vista a não só a futuros desenvolvimentos, mas também para colmatar pontos fracos actualmente existentes.

De momento, o maior ponto fraco aparenta ser a robustez mecânica da maioria dos elementos da comunidade, tal como foi notório durante a competição. Note-se que sem esta componente nem faz sentido desenvolver *software* minimamente sofisticado, uma vez que depois o *hardware* não terá capacidade de dar a devida resposta. No entanto, este factor poderá ser colmatado quando a equipa passar a dispor de uma equipa de plataformas comerciais idênticas à única já existente.

Sendo assim, este capítulo foca-se mais nas evoluções ao nível do *software*. As ideias apresentadas são: implementação faseada de um gerador automático de código para construir maquinas de estado especificas para este problema, refinação da estrutura de dados *blackboard* (ainda não completamente um tipo abstracto de informação), ideias a implementar quando a capacidade de auto-localização estiver disponível.

6.1. Gerador automático de código para máquinas de estado

O núcleo da aplicação é uma máquina de estados implementada em linguagem C com um ciclo *switch... case*. Assim, é possível identificar estados e transições de estado. Com o objectivo de facilitar o desenvolvimento de uma aplicação desta natureza poderia-se desenvolver uma ferramenta que a partir de uma IDL (*interface language*) a especificar, poderia numa fase inicial, gerar o código C referente à estrutura base da máquina de estados especificada no ficheiro IDL. Esse código seria a construção automática das definições dos estados possíveis (um enumerado); a função de despacho que seria a estrutura ciclo *switch... case*, que ficaria em aberto para o programador inserir o código correspondente às transições de estados; o código, também em aberto das funções que cada estado invocaria e que correspondem às acções realizadas durante o esse mesmo estado.

Os objectivos de uma abordagem deste género são:

1. Obrigar a que sejam instauradas normas para o desenvolvimento do *software* da aplicação da máquina de estados, e no limite para toda a aplicação. Neste caso as normas seriam aquelas que fossem especificadas para a IDL. Desta forma passar-se-ia a ter um código com uma estruturação *standard* bem definida, uma vez que a base seria gerada automaticamente pelo gerador de código, implementado para seguir esse mesmo *standard*. Este facto pode por vezes parecer pouco importante, mas pode ser fundamental para garantir a continuidade de projectos de longo prazo no qual

intervêm várias pessoas ao longo do tempo. Assim seria muito mais fácil gerir a entrada e saída de pessoas, ao contrário do modelo em que existe um programador que implementa as aplicações segundo o seu estilo muito próprio e que para o mesmo pode ser bastante intuitivo e simples, mas para uma outra pessoa esse mesmo estilo já poderá ser completamente indecifrável.

2. Evoluir sucessivamente através do refinamento dessa IDL para uma aplicação mais sofisticada onde eventualmente se possam definir não só os estados mas as transições de estados, etc. Também se poderia refinar sucessivamente a IDL e implementa-la segundo uma sintaxe e semântica já conhecida. Desta forma o projecto poderia seguir um caminho evolutivo, onde primeiro se implementariam funcionalidades básicas e simples mas que sucessivamente iriam sendo refinadas e também se onde se iriam acrescentando outras novas, como por exemplo referencias a objectos do *blackboard*, etc. Assim, em vez de se seguir uma abordagem de raiz muito complexa e sofisticada que seria muito complicada de por em funcionamento de uma só vez, entraria-se numa abordagem que de se ir sucessivamente desenvolvendo algo do simples para o complexo, à medida que se fosse ganhando maturidade nas várias fases do projecto.

3. Facilitar a construção de máquinas de estado para fins de teste e experiência. Por exemplo, para um certo problema eram propostas duas soluções de implementação. Sendo o código referente à estrutura base gerado automaticamente o programador estaria liberto para apenas se preocupar em implementar as funcionalidades. Além disso seria bastante trivial substituir o módulo da maquina de estados por uma outra uma vez que esse módulo teria independência ao resto da aplicação.

Esta aplicação poderia ser implementada recorrendo às ferramentas *lex* (para a análise lexical) e *yacc* (para a análise sintáctica) disponíveis em vários sistemas operativos incluindo o *linux*. Assim, começara-se por definir a sintaxe da IDL. Sendo uma IDL fundamentalmente declarativa, poderá utilizar-se uma sintaxe de uma linguagem conhecida que se insira nesta filosofia. No entanto, uma sintaxe deste género deverá garantir a independência com as funcionalidades da mesma, para que muito facilmente seja possível (de preferencia automaticamente) alternar para outro tipo de sintaxe que contenha as mesmas funcionalidades. Pode parecer complicado, mas utilizando devidamente as ferramentas já referidas este passo pode ser bastante pacifico e natural. Assim, poderia por exemplo definir uma IDL onde se usava uma sintaxe idêntica ao C e, automaticamente alternar para uma sintaxe idêntica ao Lisp (mas com as mesmas funcionalidades), quando a evolução do projecto assim o justificasse.

Um exemplo típico muito simples de um ficheiro IDL será algo do género do exemplo apresentado de seguida (num estilo idêntico ao da linguagem C). Note-se que existe uma estrutura especial chamada *main* e que neste caso seria a estrutura que conteria a informação referente aos estados propriamente ditos. Também o estado inicial é definido através do operador = na estrutura *main* já referida. Estas opções são apenas exemplificativas, não tendo por detrás qualquer filosofia base.

```
struct posxy
{       int     x,y;
};
void segue_bola(posxy posição_bola);
void procura_bola();
enum estados   {segue_bola, procura_bola};

struct main
{ estados      declarações = procura_bola;
};
```

Neste exemplo muito simples teríamos apenas declaração dos estados possíveis. O código gerado automaticamente para esta IDL seria algo do género:

```
/* este codigo e' apenas o esqueleto de uma maquina de estados, e serve
   para facilitar a construcao da mesma */

typedef enum estados_en       {segue_bola, procura_bola } estado;

typedef struct posxy_rec
{       int     x,y;
}* posxy;

estado segue_bola_fun(posxy arg)
{       estado ret;
        /* inserir codigo referente a este estado (segue_bola) */
        return ret
}

estado procura_bola_fun()
{       estado ret;
        /* inserir codigo referente a este estado (procura_bola) */
        return ret
}


void funcao_despacho(estado actual){
        /*      esta funcao contem o esqueleto da implementacao
                de uma maquina de estados.
        */
        posxy arg1=malloc(sizeof(struct posxy_rec));
        for(;;)
        {       switch (actual)
                {       case segue_bola: actual=segue_bola_fun(arg1);
                                break;
                        case procura_bola: actual=procura_bola_fun();
                                break;
                }
        }
}
```

Como é obvio neste ponto apenas se definem estados mas uma evolução obvia seria dotar a IDL da declaração de transições de estados, e às quais seria também gerado código. Mas essas opções terão de ser fruto de um estudo bastante aprofundado das funcionalidades que se pretendem. No entanto como se torna evidente neste exemplo muito, mas muito simples, o código segue uma normalização evidente em termos de estruturação. É este também um dos principais objectivos a atingir.

Apresenta-se agora um exemplo de uma IDL já mais sofisticada que permita declarar também as transições entre estados (num estilo de gosto meramente pessoal), bem como declarações externas e assinar uma variável a um elemento externo:

```
EXTERNO
        blackboard;
DECLARA
        posicao:        (x = blackboard.socrob[].posx,
                         y = blackboard.socrob[].posy)
ESTADOS
        procura_bola(posicao);
        segue_bola()
INICIAL
        procura_bola
TRANSICOES
        procura_bola:
                QUANDO blackboard.socrob[].seen_ball? = 1 TRANSITA segue_bola;

        segue_bola:
                QUANDO blackboard.socrob[].seen_ball? = 0 TRANSITA procura_bola;
FIM.
```

O esqueleto de código que esta IDL gerará não é apresentado mas não fugirá muito à proposta anteriormente apresentada.

6.2. Refinação da estrutura de dados *blackboard*

Os agente comunicam entre si através de uma estrutura de dados chamada *blackboard*. Consequentemente é através dessa estrutura que se baseia a cooperação entre eles, uma vez que é através da comunicação que se podem estabelecer mecanismos de cooperação. A ideia base do *blackboard* é simples: não é mais do que um repositório de variáveis de estado globais referentes a cada agente e á comunidade. Os nomes dessas variáveis seguem uma estrutura hierárquica (por exemplo: .socrob1.motores.controladores.kp). A cada variável pode também estar associada uma função que é activada quando esta é acedida. Uma especificação mais detalhada do *blackboard* encontra-se em [4].

Esta estrutura de dados já se encontrava disponível, fisicamente representada sobre a forma de uma lista. Posteriormente foi evoluída para uma tabela de dispersão implementada por nós. No entanto, o *blackboard* deve ser encarado como um tipo abstracto de informação, independente da sua estrutura física, onde é fornecida uma estrutura de dados abstracta dispondo de um conjunto de operações para a manipular. Para além disso, o *blackboard* até pela sua definição não é mais que um repositório de informação, ou um sistema de base de dados muito simplista. A característica que torna o *blackboard* uma estrutura mais interessante de desenhar é a distribuição, ou seja, este pode (e deve) estar distribuído por um conjunto de máquinas interligadas entre si por um qualquer meio de transmissão da informação. Agora, no nível mais

alto de abstracção esse distribuição física (e logo a sua estrutura física) deve estar escondida e não explicita na declaração dessas mesmas variáveis, senão deixamos de ter um tipo abstracto de informação para termos uma estrutura de dados física especifica. Além disso a algorítmica da aplicação deverá ser completamente independente a estrutura de dados. Podendo uma variável do *blackboard* ser caracterizada por propriedades que tem relacionamento directo com a sua estrutura física então termos uma estrutura pouco flexível e em que uma alteração na estrutura física desta terá repercussões nos algoritmos da aplicação.

O *blackboard* é distribuído, mas poderá e deverá criar-se uma camada de abstracção que dê a ilusão que se está a trabalhar com uma estrutura centralizada. Assim, não fará sentido falar em termos de localização (*location*) da variável, pelo menos em termos do alto nível. Será no baixo nível que é tomado conhecimento se uma variável reside numa máquina local ou remota. Este modelo é inspirado no mecanismo de gestão de memória *Mach* [21]. O *Mach* foi desenvolvido inicialmente como uma variante do sistema BSD. A gestão de memória é efectuada com base em objectos de memória que em termos conceptuais são um bloco contínuo de memória virtual, que pode ser reservado no espaço de endereçamento de um processo. O *Mach* introduz o conceito inovador ao permitir que as faltas de página possam ser tratadas por um processo fora do núcleo (*external pager*). Ou seja quando existe uma falta de página é esse processo que decide onde a ir buscar. Com as devidas diferenças parece ser uma filosofia boa para o nosso caso. Desta forma o utilizador do nível de abstracção mais elevado não tem de se preocupar com a disposição física dos dados. Para lhes aceder existe um conjunto de operações e são essas operações que o utilizador terá de conhecer e qual o efeito de invocar uma dessas operações sobre o tipo de dados.

Assim, uma referência para um elemento do *blackboard* seria no baixo nível um ponteiro mais refinado: um triplo (máquina, porto, endereço). A máquina seria uma referência à máquina onde residiria a variável do *blackboard* em questão. O porto seria a referência ao processo onde essa variável estaria acessível, desta forma é possível haver mais que um processo na mesma máquina onde cada um teria um bocado do *blackboard*, desta forma cada processo terá de ter associado a si um porto para comunicação com o exterior. O endereço seria o ponteiro na sua definição usual, ou seja, o endereço de memória dentro do espaço de endereçamento do processo respectivo onde a variável do *blackboard* reside fisicamente.

Desta forma deixará de fazer sentido falar em termos de alcance das variáveis, *global* ou *local*. Esse detalhe é uma característica da representação física, e o *blackboard* quer-se um tipo abstracto de informação. Ou seja, a decisão da melhor localização da variável passará a ser da responsabilidade do baixo nível e não interessa ao utilizador do tipo de dados. A este apenas interessa a sintaxe e semântica

das operações disponíveis para aceder ao *blackboard*. Faça-se um paralelismo com as arquitecturas de computadores de memória distribuída. Imagine-se um computador com 8 placas, cada uma destas com a sua memória, estando estas ligadas entre si por um barramento. É da responsabilidade do *hardware* garantir a transparência da arquitectura em relação a uma arquitectura de memória partilhada: para o utilizador comum não lhe interessa saber se as páginas de memória reservadas para a sua aplicação residem na mesma placa do processador que executa a mesma. Claro que por motivos de eficiência é de aconselhar que o *hardware* seja suficientemente inteligente para tomar este tipo de decisões, mas estas não interessam minimamente ao utilizador. Mais, não fará qualquer sentido o utilizador ter de declarar nas suas aplicações onde quer que sejam reservadas as páginas de memória que vai utilizar: se locais à memória da placa do processador utilizado, se remotas.

O *blackboard* sendo conceptualmente uma estrutura de dados centralizada, também não faz sentido falar em termos de *policy* como sendo *broadcast* ou *indexed* uma vez que havendo um só *blackboard*, e não um conjunto destes, não faz sentido falar em termos de variáveis que são actualizados nos outros *blackboards* locais ou nos remotos. No entanto poderá fazer sentido haver variáveis que estejam no espaço físico de mais que uma máquina. Este mecanismo poderá ser implementado fazendo a ilusão de que só existe um *blackboard* centralizado, mas cada máquina possui uma *cache* onde será guardada a informação mais recentemente ou frequentemente acedida. Para implementar o mecanismo de *broadcast* define-se um primitiva *cache-update* que faz uma espécie de actualização das caches locais. Ou então definir as propriedades da *policy* como sendo *updated* ou *non-updated*, ou seja quando a variável é *updated* a máquina que contém a variável terá de avisar as outras máquinas que a variável x foi modificada, logo terão de actualizar as suas *cahes*. O nome *cache* parece ser um nome interessante uma vez que representa uma zona de memória de acesso mais eficiente, mas que pode não estar sempre consistente com a informação correspondente que existe na memória central (neste caso o *blackboard*). Para o efeito terá de se implementar um protocolo de coerência de caches. Numa fase inicial, um protocolo simples inspirado no *dragon*, existente em algumas arquitecturas de multiprocessadores poderá ser a opção mais indicada. O protocolo *dragon* é bastante simples: sempre que um processador (neste caso um agente) modifica uma variável presente na sua *cache* de dados, executa uma difusão do novo valor para que os outros processadores actualizem (caso seja necessário) as suas *caches* com o valor respectivo.

Estas ideias são inspirada nas arquitecturas de multiprocessador de memória centralizada, como a descrita em [22], embebida no mecanismo de gestão de memória do *Mach* [21].

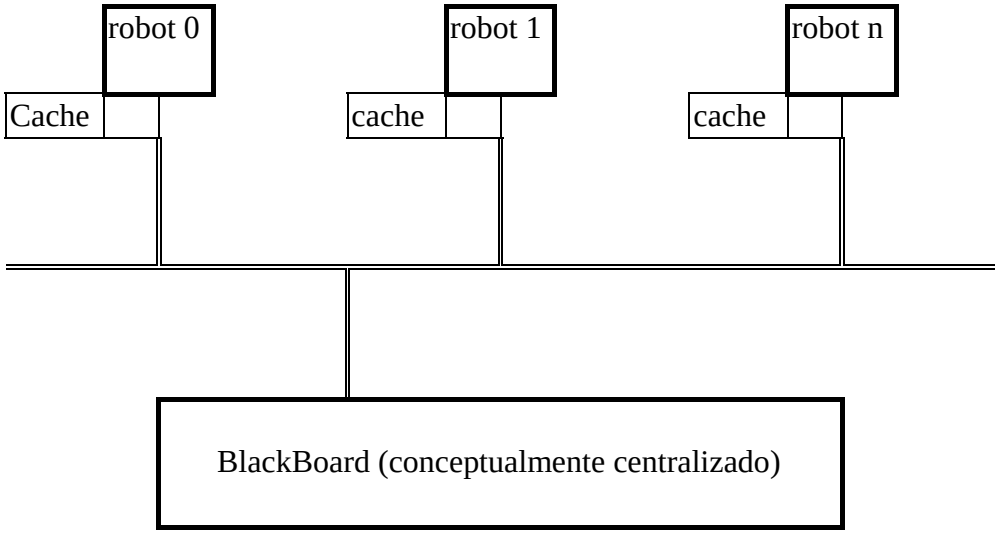


Figura 6.1.: arquitectura conceptual do sistema do BB centralizado

As eventuais primitivas de sincronização juntamente com a propriedade *lock* não fazem sentido existirem no alto nível. Primeiro pela definição de alto nível, não parece ser boa política fornecer o acesso a recursos de um sistema operativo específico. Segundo por uma questão de robustez: não é decididamente seguro estar a proteger e desproteger variáveis remotamente. Se localmente já tem de se prover os sistemas de uma grande complexidade, para garantir que uma certa quantidade de mecanismos tem de decorrer atomicamente [22], a complexidade que não será quando se migra para um ambiente distribuído. A ideia deverá ser de que a sincronização é feita localmente e no baixo nível: transparente para o programador. Ou seja, um dado robot x só tem de enviar uma ordem de modificação de uma variável a um outro robot y, e é dentro do robot y que são feitos os *locks* e os *unlocks* necessários sem que o robot x ou o programador saiba do que se passou e como se passou. Agora imagine-se que um robot envia uma ordem para actualização de uma variável para outro: o robot onde essa variável reside fisicamente. Passado um instante microscopicamente muito próximo outro robot enviaria também uma mensagem de actualização da mesma variável ao mesmo robot, mas que chegaria primeiro (muitíssimo provável de acontecer com os *modems* utilizados). Neste caso não seria preservada a sequêncialidade temporal das operações. Nesta fase, a melhor solução para este problema seria assumir que a mensagem que chegou primeiro foi aquela que também foi enviada primeiro, uma vez que ao nível macroscópico (e os robots funcionam a esse nível) elas foram enviadas ao mesmo tempo.

6.3. Ideias a implementar quando o mecanismo de auto-localização estiver disponível

Uma das ideias avançadas durante o projecto será a de integrar na interface, no módulo de inicio de jogo a funcionalidade de ser possível deslocar automaticamente todos os agentes para uma posição especifica do campo. Essa capacidade pode ser facilitada possibilitando ler esses valores de um ficheiro. Assim, poderiam-se construir ficheiros de inicio (ou de outras situações de treino) de jogo.

Também se poderia construir uma janela da interface gráfica com o campo desenhado, onde se observaria em tempo real a posição de cada jogador dentro do mesmo. Esta funcionalidade é de interesse elevado uma vez que possibilita de uma forma muito imediata avaliar se um determinado robot se está a auto-localizar correctamente.

Acima de tudo a auto-localização abrirá caminho a que se implementem um conjunto muito interessante de funcionalidades, nomeadamente ao nível da cooperação entre os agentes, como por exemplo, ter zonas fixas do campo para cada um, mais facilmente decidirem que agente deverá ir à bola para atacar a baliza, etc.

7. CONCLUSÃO SOBRE O TRABALHO DESENVOLVIDO

O trabalho desenvolvido proporcionou uma experiência muito enriquecedora no que diz respeito ao desenvolvimento de um projecto no qual existe uma equipa de desenvolvimento algo heterogénea com membros de várias áreas. Torna-se notório que uma situação é desenvolver aplicações típicas de um projecto de uma cadeira da licenciatura onde intervém dois ou três alunos do mesmo ramo, e participar num projecto desta natureza, também ao nível da gestão dos recursos humanos e materiais. Para além disso foi possível contactar pela primeira vez com algum material inédito para os alunos. Não esquecendo de referir a experiência de participar num evento académico internacional onde se esteve em contacto directo com outras metodologias de trabalho e soluções técnicas.

A principal aprendizagem (também à custa dos erros cometidos) foi ao nível da necessidade da criação de uma disciplina interna para ser possível o desenvolvimento de um projecto onde intervêm não uma ou duas pessoas muito próximas, mas uma equipa de desenvolvimento.

Outra conclusão é a importância de num projecto desta natureza, com características de inovação, em desenhar uma arquitectura na qual os vários módulos sejam organicamente independentes entre si. Esta condição pode tornar-se de primeira linha uma vez que estando sucessivamente o projecto em evolução é relativamente comum realizarem-se alterações, melhorias, etc., na arquitectura, e se existir uma rede de dependências entre vários módulos, uma pequena alteração num modulo pode ter repercussões negativas por todos os outros. É assim fundamental para além de tentar apenas preservar as dependências mínimas, garantir a independência entre estruturas de dados e algoritmos.

Para além disso, tendo sido o trabalho integrado num projecto de longo prazo desenvolvido por várias pessoas ao longo do tempo, foi uma experiência muito útil no que diz respeito à sensibilização a aspectos como metodologia e normalização do trabalho desenvolvido de forma a ser possível um entendimento entre os vários intervenientes. Também para ser possível que muito trabalho desenvolvido seja aproveitado no futuro e que não caia em desuso apenas porque os novos elementos não atingem aquilo que foi desenvolvido no passado.

8. BIBILOGRAFIA

[1] Robocup Organization. URL:http://www.robocup.org

[2] Pedro Lima, Rodrigo Ventura, Pedro Aparício, Luis Custódio. "A Functional Architecture for a Team of Fully Autonomous Cooperative Robots", Proc. of RoboCup Workshop of IJCAI 99.

[3] RodrigoVentura, Pedro Aparício, Pedro Lima. "RoboCup98 ISocRob Team Technical Reference", ISR Internal Report RT-702-99 and RT-404-99 July 1999.

[4] Rodrigo Ventura, Pedro Aparício, Pedro Lima. "Agent-Based Programming Language for Multi-Agent Teams", ISR Internal Report RT-701-99, RT-401.

[5] Pedro Aparício, RodrigoVentura, PedroLima, Carlos Pinto-Ferreira."ISocRob - Team Description", In Minoru Asada and Hiroaki Kitano, editors, RoboCup-98: Robot Soccer World Cup II, Springer-Verlag, Berlin, 1999.

[6] Rodrigo Ventura, Pedro Aparício, Pedro Lima, Carlos Pinto-Ferreira. "SocRob - A Society of Cooperative Mobile Robots", Proc. of 1998 IEEE International Conference on Systems, Man, and Cybernetics.

[7] Linux Online. URL: http://www.linux.org

[8] Bradford Nichols, Dick Buttlar & Jacqueline Proulx Farrel. *Pthreads Programming,* O'Reilly,1998.

[9] Welch, Brent B. – *Pratical Programing in Tcl and Tk*. Nova Jersey: Prentice Hall, 1997. 2ed ISBN 0-13-616830-2.

[10] Ousterout,John. "Tcl and the Tk Toolkit".

[11] "Scriptics". URL: http://www.scriptics.com

[12] "Scriptics Tcl Consortium". URL: http://www.tclconsortium.com

[13] "Tcl/Tk Repository". URL:http://www.tcltk.com

[14] M.Woodridge, N.Jennings. *Intelligent Agents: Theory and Pratice.*

[15] Silva, Alberto. *Agentes de Software na Internet*. Edições Centro Atlântico,1999.

[16] José Alves Marques, Paulo Guedes. *Fundamentos de Sistemas Operativos*, Editorial Presença,2ªedição,1992.

[17] Oliver Jones. *Introduction to the X Window System.* Prentice Hall,1995.

[18] Roger S. Pressman. *Software Engineering: Apractitioner's Approach*, McGraw-Hill, Third Edition,1994.

[19] "Users's Manual", Nomadic Techonologies inc.

[20] Nomadic Technologies, Inc. URL: http://www.robots.com

[21] M. "Mach: A New Foundation for Unix Development", Proc. Summer Usenix, Julho 1986, pág; 93-112.

[22] J. Hennessy, D. Patterson "Computer Architecture, a Quantitative Approach", Morgan Kaufmann Publishers, Inc., 1996, pág 654-676, pág 694-707.

APÊNDICE A, PEQUENOS PROGRAMAS DE TESTE

Neste anexo, são apresentados pequenos programas implementados em linguagem C, ambiente *Linux*, e que são exemplificativos da utilização das soluções avançadas para os problemas descritos acima. É também apresentada uma pequena explicação informal sobre o funcionamento dos mesmos.

Estes pequenos programas foram desenvolvidos aquando da implementação da aplicação global, e tinham como objectivo testar as funcionalidades e soluções especificadas, por forma a ser possível decidir até que ponto algumas soluções apresentadas eram viáveis na prática. Desta forma, através destes programas de teste é também possível observar a evolução do projecto ao longo do tempo uma vez que estes representam de uma forma simplista os módulos principais da aplicação.

Secção A1

Exemplo da criação de dois processos em *Linux* usando a primitiva *fork*.

```
#include <unistd.h>
#include <stdio.h>
#include <errno.h>

extern int errno;

int main()
{ int pid,pid_term,estado;

  pid=fork();

  /* fork() executado sem sucesso, escreve causa no stderr e termina */
  if(pid==-1)
  { printf("<pai>  : comando fork() executado sem sucesso\n");
    perror("causa  ");
    exit(-1);
  }

  /* processo filho */
  if(pid==0)
  { printf("<filho>: acabei de ser criado com PID = %d\n",getpid());

    /* inserir aqui codigo do processo filho */

    exit(1);
  }
  /* processo pai */
  else
  { printf("<pai>  : processo filho criado com PID = %d\n",pid);

    /* inserir aqui codigo do processo pai */

    /* espera que o processo filho termine, exibe mensagem e termina */
    pid_term=wait(&estado);
    printf("<pai>  : terminou o processo com PID = %d com o estado %d\n",
           pid_term,estado);
    exit(1);
  }
}
```

Exemplo A1: ficheiro e_fork.c

Este exemplo, mostra a criação de um processo através da primitiva *fork*. Este primitiva cria um processo filho que apenas difere do processo pai no seu PID (*process identification*) e PPID (*parent process identification*). Além disso, as utilizações de recursos são actualizadas a 0, os trincos de ficheiros (*file locks*) e os *signals* pendentes não são herdados.

Em *Linux* a primitiva *fork* é utilizada usando copias-em-escrita (*copy-on-write*) sobre páginas, levando a que a única penalização de uma invocação a esta primitiva seja o tempo e a memória requerida para duplicar as tabelas de páginas do processo pai, e para criar uma estrutura de tarefas única para o filho.

Em caso de sucesso, o PID do processo filho é retornado para o processo pai, e o valor 0 é retornado para o processo filho. Em caso de falha na criação do processo filho, a primitiva devolve -1, e a variável *errno* é actualizada com o valor apropriado.

Secção A2

Exemplo da criação de duas tarefas em *Linux* interiores ao mesmo processo.

```
#include <pthread.h>
#include <stdio.h>

struct par
{ int int1,int2;
};

void tar1(struct par *p)
{ sleep(1);
  printf("a soma dos elementos: %d\n",p->int1+p->int2);
}

void tar2(struct par *p)
{ p->int1=2;
  p->int2=4;
}


void main()
{ struct par p;
  pthread_t tarefa1,tarefa2;

  p.int1=p.int2=0;
  pthread_create(&tarefa1,NULL,(void*)tar1,&p);
  pthread_create(&tarefa2,NULL,(void*)tar2,&p);

  pthread_join(tarefa1,NULL);
  pthread_join(tarefa2,NULL);
}
```

Exemplo A2: ficheiro e_thread.c

Secção A3

Exemplo de utilização de um *double buffer* de memória partilhada entre dois processos *Linux*, recorrendo a sincronismo.

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

#if defined(__GNU_LIBRARY__) && !defined(_SEM_SEMUN_UNDEFINED)
/* union semun e' definida incluindo <sys/sem.h> */
#else
/* de acordo com o X/OPEN teremos que a definir manualmente*/
union semun
{ int val;                    /* value for SETVAL */
  struct semid_ds *buf;       /* buffer for IPC_STAT, IPC_SET */
  unsigned short int *array;  /* array for GETALL, SETALL */
  struct seminfo *__buf;      /* buffer for IPC_INFO */
};
#endif

void erro(char *s)
{ printf(s);
  exit(1);
}

/* cria um par mutex, semaforo */
void inisem(int nmax,int semmutex,int semaloc,int *mutex,int *aloc)
{ union semun init;
  if((*mutex=semget(semmutex,1,0777|IPC_CREAT))<0)
    erro("erro semget (semmutex)\n");
  if((*aloc=semget(semaloc,1,0777|IPC_CREAT))<0)
    erro("erro semget (semaloc)\n");
  init.val=1;
  if(semctl(*mutex,0,SETVAL,init)<0)
    erro("erro semctl (semmutex)\n");
  init.val=nmax;
  if(semctl(*aloc,0,SETVAL,init)<0)
    erro("erro semctl (semaloc)\n");
}

/* acede a um par mutex, semaforo ja' criados */
void abresem(int nmax,int semmutex,int semaloc,int *mutex,int *aloc)
{ union semun init;
  if((*mutex=semget(semmutex,1,0777|IPC_EXCL))<0)
    erro("erro semget (semmutex)\n");
  if((*aloc=semget(semaloc,1,0777|IPC_EXCL))<0)
    erro("erro semget (semaloc)\n");
}

/* testa o mutex, bloqueia ate' este ser libertado se for caso disso */
void esperarmutex(key_t semid,int semnum)
{ struct sembuf s;
  s.sem_num=semnum;
  s.sem_op=-1;
  s.sem_flg=SEM_UNDO;
  if(semop(semid,&s,1)<0)
    erro("erro semop (esperarmutex)\n");
}

/* liberta o mutex */
void assinalarmutex(key_t semid,int semnum)
{ struct sembuf s;
  s.sem_num=semnum;
  s.sem_op=1;
  s.sem_flg=SEM_UNDO;
  if(semop(semid,&s,1)<0)
    erro("erro semop (assinalarmutex)\n");
}

/* testa o semaforo, bloqueia ate' este ser libertado se for caso disso */
void esperar(key_t semid,int semnum,int uni)
{ struct sembuf s;
  s.sem_num=semnum;
  s.sem_op=-uni;
  s.sem_flg=0;
  if(semop(semid,&s,1)<0)
    erro("erro semop (esperar)\n");
}
```

```
/* liberta o semaforo */
void assinalar(key_t semid,int semnum,int uni)
{ struct sembuf s;
  s.sem_num=semnum;
  s.sem_op=uni;
  s.sem_flg=0;
  if(semop(semid,&s,1)<0)
    erro("erro semop (assinalar)\n");
}
```

Exemplo A3.1: sem.c

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <signal.h>
#include "sem.c"

#define CHAVEMEM0 10        /* identificador da chave para acesso ao buffer 0 */
#define CHAVEMEM1 11        /* identificador da chave para acesso ao buffer 1 */
#define TAMMEM    6         /* espaco de memoria alocada */
#define NMAX      1         /* numero de inicializacao dos semaforos */
#define SEMMUTEX0 5         /* identificador do mutex de acesso ao buffer 0 */
#define SEMALOC0  6         /* identificador do semaforo de acesso ao buffer 0 */
#define SEMMUTEX1 7         /* identificador do mutex de acesso ao buffer 1 */
#define SEMALOC1  8         /* identificador do semaforo de acesso ao buffer 1 */
#define FLAG      0777      /* flag dos semaforos */
#define TIME      2         /* tempo que demora a produzir um elemento */

int *shmp0,shmid0,*shmp1,shmid1;

void sair()
{ printf("\nA: Terminando o processo produtor\n");
  shmctl(shmid0,IPC_RMID,NULL);
  shmctl(shmid1,IPC_RMID,NULL);
  exit(0);
}

int main()
{ int i,n0,n1,mutex0,semaloc0,mutex1,semaloc1;
  char b=0;

  signal(SIGINT,sair);

  /* inicia os pares mutex, semaforo */
  inisem(NMAX,SEMMUTEX0,SEMALOC0,&mutex0,&semaloc0);
  inisem(NMAX,SEMMUTEX1,SEMALOC1,&mutex1,&semaloc1);

  /* inicia a memoria partilhada */
  if((shmid0=shmget(CHAVEMEM0,TAMMEM*sizeof(int),FLAG|IPC_CREAT))<0)
    erro("A: erro shmget (regiao 0)\n");
  printf("A: Regiao 0 criada com a chave: %d\n",CHAVEMEM0);
  if((shmid1=shmget(CHAVEMEM1,TAMMEM*sizeof(int),FLAG|IPC_CREAT))<0)
    erro("A: erro shmget (regiao 1)\n");
  printf("A: Regiao 1 criada com a chave: %d\n\n",CHAVEMEM1);
  if((shmp0=(int*)shmat(shmid0,(char*)0,0))==(int*)-1)
    erro("A: erro shmat (regiao 0)\n");
  if((shmp1=(int*)shmat(shmid1,(char*)0,0))==(int*)-1)
    erro("A: erro shmat (regiao 1)\n");

  /* ciclo de producao */
  for(n0=0,n1=50,b=!b;;)
  {
    if(b)
    { /* produz buffer 0 */
      esperar(semaloc0,0,1);
      esperarmutex(mutex0,0);
      printf("A: Regiao 0: X varia entre  0 e 49 (produzindo...)\n");
      printf("A: Regiao 0:");
```

```
      for(i=0;i<TAMMEM;i++)
      { shmp0[i]=n0;
        printf("%3d",shmp0[i]);
        fflush(stdout);
        sleep(TIME);
      }
      assinalarmutex(mutex0,0);
      n0++;
    }
    else
    { /* produz buffer 1 */
      esperar(semaloc1,0,1);
      esperarmutex(mutex1,0);
      printf("A: Regiao 1: X varia entre 50 e 99 (produzindo...)\n");
      printf("A: Regiao 1:");
      for(i=0;i<TAMMEM;i++)
      { shmp1[i]=n1;
        printf("%3d",shmp1[i]);
        fflush(stdout);
        sleep(TIME);
      }
      assinalarmutex(mutex1,0);
      n1++;
    }

    /* troca de buffers */
    b=(!b);

    printf("\n\n");
    if(n0>=49) n0=0;
    if(n1>=99) n1=50;
  }
  return 0;
}
```

Exemplo A3.2: e_mempA.c

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <signal.h>
#include "sem.c"

#define CHAVEMEM0 10        /* identificador da chave para acesso ao buffer 0 */
#define CHAVEMEM1 11        /* identificador da chave para acesso ao buffer 1 */
#define TAMMEM    6         /* espaco de memoria alocada */
#define NMAX      1         /* numero de inicializacao dos semaforos */
#define SEMMUTEX0 5         /* identificador do mutex de acesso ao buffer 0 */
#define SEMALOC0  6         /* identificador do semaforo de acesso ao buffer 0 */
#define SEMMUTEX1 7         /* identificador do mutex de acesso ao buffer 1 */
#define SEMALOC1  8         /* identificador do semaforo de acesso ao buffer 1 */
#define FLAG      0777      /* flag dos semaforos */
#define TIME      1         /* tempo que demora a consumir um elemento */

int *shmp0,shmid0,*shmp1,shmid1;

void sair()
{ printf("\nB: Terminando processo consumidor\n");
  (void)shmdt(shmp0);
  (void)shmdt(shmp1);
  exit(0);
}

int main()
{ int i,mutex0,semaloc0,mutex1,semaloc1;
  char b=0;

  signal(SIGINT,sair);

  /* acede aos pares mutex, semaforo */
```

```
  abresem(NMAX,SEMMUTEX0,SEMALOC0,&mutex0,&semaloc0);
  abresem(NMAX,SEMMUTEX1,SEMALOC1,&mutex1,&semaloc1);

  /* acede a memoria partilhada ja' criada */
  if((shmid0=shmget(CHAVEMEM0,TAMMEM*sizeof(int),FLAG))<0)
    erro("B: erro shmget (regiao 0)\n");
  printf("B: Entrando na regiao 0 com a chave: %d\n",CHAVEMEM0);
  if((shmid1=shmget(CHAVEMEM1,TAMMEM*sizeof(int),FLAG))<0)
    erro("B: erro shmget (regiao 1)\n");
  printf("B: Entrando na regiao 1 com a chave: %d\n\n",CHAVEMEM1);
  if((shmp0=(int*)shmat(shmid0,(char*)0,0))==(int*)-1)
    erro("B: erro shmat (regiao 0)\n");
  if((shmp1=(int*)shmat(shmid1,(char*)0,0))==(int*)-1)
    erro("B: erro shmat (regiao 1)\n");

  /* ciclo de consumo */
  for(;;)
  {
    if(!b)
    { /* consome buffer 0 */
      esperarmutex(mutex0,0);
      printf("B: Regiao 0: X varia entre  0 e 49 (consumindo...)\n");
      printf("B: Regiao 0:");
      for(i=0;i<TAMMEM;i++)
      { printf("%3d",shmp0[i]);
        fflush(stdout);
        sleep(TIME);
      }
      assinalarmutex(mutex0,0);
      assinalar(semaloc0,0,1);
    }
    else
    { /* consome buffer 1 */
      esperarmutex(mutex1,0);
      printf("B: Regiao 1: X varia entre 50 e 99 (consumindo...)\n");
      printf("B: Regiao 1:");
      for(i=0;i<TAMMEM;i++)
      { printf("%3d",shmp1[i]);
        fflush(stdout);
        sleep(TIME);
      }
      assinalarmutex(mutex1,0);
      assinalar(semaloc1,0,1);
    }

    /* troca de buffers */
    b=(!b);

    printf("\n\n");
  }
  return 0;
}
```

Exemplo A3.2: e_mempB.c

Secção A4

Exemplo de troca de *signals* entre dois processos em *Linux*.

```
#include <signal.h>
#include <stdio.h>

int pidA,pidB;
```

```
void usrA()
{ signal(SIGUSR1,usrA);
  printf("pA: SIGUSR1 (recebido)\n");
  printf("pA --- SIGUSR2 --> pB\n");
  kill(pidB,SIGUSR2);
}

void usrB()
{ signal(SIGUSR2,usrB);
  printf("pB: SIGUSR2 (recebido)\n");
}

void sairA()
{ printf("pA: terminando execucao\n");
  printf("pA --- SIGINT --> pB\n");
  kill(pidB,SIGINT);
  exit(0);
}

void sairB()
{ printf("pB: terminando execucao\n");
  printf("pB --- SIGINT --> pA\n");
  kill(pidA,SIGINT);
  exit(0);
}

void main()
{ pidB=getpid();
  pidA=fork();

  if(pidA==0)
  { signal(SIGUSR1,usrA);
    signal(SIGINT,sairA);
    for(;;)
      pause();
  }
  else
  { signal(SIGUSR2,usrB);
    signal(SIGINT,sairB);
    for(;;)
    { sleep(2);
      printf("pB --- SIGUSR1 --> pA\n");
      kill(pidA,SIGUSR1);
      pause();
    }
  }
}
```

Exemplo A4: e_signal.c

Para se enviar explicitamente um *signal* (evento assíncrono) a um processo recorre-se à primitiva *kill*. O nome da primitiva é infeliz, uma vez que a primitiva não tem obrigatoriamente como efeito a terminação do processo à qual é pedido o envio do *signal*. Assim, para um processo poder enviar um *signal* a outro tem de conhecer o seu PID (*process identification*). Para interceptar os *signals* existentes, recorre-se à primitiva *signal* que substitui a rotina associada por defeito ao *signal* correspondente pela rotina explicitada pelo utilizador.

Existe um pequeno pormenor muito traiçoeiro associado à intercepção dos *signals* que não será aqui abordado com detalhe. Um *signal* apenas é interceptado para uma invocação. Após esta ter ocorrido, é-lhe reposta a rotina por defeito. Se se quiser que o *signal* continue associado à rotina do programador é necessário interceptar novamente o *signal*. No exemplo acima, assume-se que as rotinas

associadas por defeito aos *signals* são repostas logo após a ocorrência do *signal* para desta forma o *signal* continuar associado à rotina. Assim, é necessário que a primeira instrução desta seja novamente a intercepção do *signal*. Esta solução não é no entanto perfeita, nomeadamente pode causar problemas caso ocorra o mesmo *signal* durante o período que decorre entre este e a sua nova intercepção. Também em situações de chamadas sucessivas à mesma rotina derivadas da ocorrência sucessiva do mesmo *signal* pode causar problemas. Aqui, dependendo da versão (e distribuição) do sistema operativo, pode ainda ocorrer a situação de o sistema operativo inibir a recepção de *signals* dentro das rotinas de tratamento. Mas como já foi referido, este pormenor não será abordado com detalhe.

Secção A5

Exemplo de utilização de *sockets* sem ligação para comunicação entre dois processos em *Linux*.

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <signal.h>

#define PORTO      20000
#define MAX_BUFFER 100

void erro(char *s)
{ printf("servidor: erro na primitiva %s\n",s);
  exit(0);
}

int main()
{ struct sockaddr_in serv_addr,cli_addr;
  char msg[MAX_BUFFER];
  int sockfd,n,cli_dim=sizeof(struct sockaddr);

  signal(SIGPIPE,SIG_IGN);

  if((sockfd=socket(AF_INET,SOCK_DGRAM,0))<0)
    erro("socket");
  bzero((char*)&serv_addr,sizeof(serv_addr));
  serv_addr.sin_family=AF_INET;
  serv_addr.sin_addr.s_addr=htonl(INADDR_ANY);
  serv_addr.sin_port=htons(PORTO);
  if(bind(sockfd,(struct sockaddr*)&serv_addr,sizeof(serv_addr))<0)
    erro("bind");

  for(;;)
  { if((n=recvfrom(sockfd,msg,MAX_BUFFER,0,&cli_addr,&cli_dim))<0)
      erro("recvfrom");
    printf("servidor > recebido: %s\n",msg);
    if(sendto(sockfd,msg,n,0,&cli_addr,cli_dim)<0)
      erro("sendto");
  }

}
```

Exemplo A5.1: e_servSock.c

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <signal.h>

#define PORTO_SERV 20000
#define PORTO_CLI  20001
#define MAX_BUFFER 100
#define NOME_SERV  "localhost.localdomain"

void erro(char *s)
{ printf("client: erro na primitiva %s\n",s);
  exit(0);
}

int main()
{ struct sockaddr_in serv_addr,cli_addr,remet_addr;
  char env_msg[MAX_BUFFER],rec_msg[MAX_BUFFER];
  struct hostent *hp;
  int sockfd,n,remet_dim=sizeof(struct sockaddr_in);

  if((sockfd=socket(AF_INET,SOCK_DGRAM,0))<0)
    erro("socket");
  bzero((char*)&cli_addr,sizeof(cli_addr));
  cli_addr.sin_family=AF_INET;
  cli_addr.sin_addr.s_addr=htonl(INADDR_ANY);
  cli_addr.sin_port=htons(PORTO_CLI);
  if(bind(sockfd,(struct sockaddr*)&serv_addr,sizeof(serv_addr))<0)
    erro("bind");

  bzero((char*)&serv_addr,sizeof(serv_addr));
  serv_addr.sin_family=AF_INET;
  if(!(hp=gethostbyname(NOME_SERV)))
    erro("gethostbyname");
  bcopy(hp->h_addr,&serv_addr.sin_addr.s_addr,hp->h_length);

  printf("string a enviar: ");
  fgets(env_msg,MAX_BUFFER,stdin);
  if(sendto(sockfd,env_msg,strlen(env_msg)+1,0,
    (struct sockaddr*)&serv_addr,sizeof(serv_addr))<0)
    erro("sendto");
  if(recvfrom(sockfd,rec_msg,MAX_BUFFER,0,
    (struct sockaddr*)&remet_addr,&remet_dim)<0)
    erro("recvfrom");
  printf("client > recebido: %s\n",rec_msg);

  return 0;
}
```

Exemplo A5.2: e_cliSock.c

Os *sockets* são uma tecnologia já bastante divulgada no meio académico pelo que se omite uma explicação sobre o mecanismo de funcionamento dos mesmos.

Secção A6

Exemplo de utilização de funções de conversão de dados da rede.

Exemplo A6: e_conv.c

Secção A7

Exemplo de utilização da solução para replicação.

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>

#define N_REPL 5

int main()
{ int i,status;

  /* inserir aqui inicializacoes */

  /* replicacao vai efectuar-se 5 vezes, terminando os processo logo apos a 5a vez */
  for(i=0;i<N_REPL;i++)
  if(fork()!=0)
  { /* processo pai */
    wait(&status);
    printf("primario: o secundario morreu\n");
    if(i==N_REPL-1)
      printf("primario: terminando\n");
    else
      printf("primario: lancando outro secundario\n");
  }
  else
  { /* processo filho */
    int n;

    printf("secundario: iniciando processamento\n");

    /* inserir codigo do processo filho */

    for(n=0;n<i*10;n++)
    { printf(".");
      fflush(stdout);
      usleep(100000);
    }
    printf("\nsecundario: falha!!! terminando de forma abruta\n");
    exit(-1);
  }
  return 0;
}
```

Exemplo A7: e_repl.c

A base desta solução reside na primitiva *wait*: uma primitiva que bloqueia o processo pai até o(s) processos filhos morrerem que foram lançados. Desta forma, o processo pai, inicializa os canais de comunicação, e através da primitiva *fork* cria um processo filho que herda os ditos canais. De seguida bloqueia-se no *wait* até o processo filho eventualmente morrer. Quando isto acontece desbloqueia-se e volta a lançar um outro processo filho, que novamente herda os canais de comunicação já inicializados. Desta forma é possível a um outro processo que comunica com estes canais, nem sequer se aperceber que ouve uma falha no sistema, uma vez que não foi

necessário recorrer à inicialização dos canais de comunicação por o processo filho ter falhado.

APÊNDICE B, DESCRIÇÃO GERAL DO ROBOT *NOMAD SUPER SCOUT II*

O *Nomad Super Scout II* é um robot móvel com um Pc industrial *onboard*, módulos de sensores constituídos por ultra-sons e *bumpers*, e um sistema de visão opcional.

Num nível superior, O *Nomad Super Scout II* é controlado por um pequeno PC industrial, o qual é montado internamente.

Computador On-board

O processador *high-level* deste sistema é um *Pentium* a 200MHZ colocado numa placa industrial. O processador *high-level* comunica com o processador *low-level* através da porta série. O processador *low-level* deste sistema é um Motorola MC68332.

Sensores

O *Nomad Super Scout II* vem de raiz com sensores de odometria, um anel com 6 sensores de contacto, 16 sensores de sonares ultra-sónicos em que o alcance dos sonares é de 15cm até 650cm.

Especificações mais relevantes

Diametro:	41 cm.
Altura:	35 cm.
Peso:	25 kg. (com as baterias)
Movimento:	velocidade: 1.0m/sec, acelaração: 2m/s2
Payload:	5 kg.
Potência da Bateria:	432 watt-hora
Sistema de Tracção:	tracção diferencial com 2 motores independentes
Ground Clearance:	1.5 cm
Processador:	*Pentium* 233 MHz + Motorola 68332-16 MHz
Disco Rígido:	2 GB
Memória:	32 MB
Portas: Porta Série:	38.4 Kbaud , porta de *joystick*
Resolução do Encoder:	Translação: 756 counts/cm,
	Rotação: 230 counts/*degree*

APÊNDICE C, CÓDIGO DA APLICAÇÃO

Ficheiros da aplicação principal

Ficheiros contendo as estruturas de dados e protótipos

ficheiro interface.h

```
#ifndef INTERFACE_H_
#define INTERFACE_H_
#define BUFSIZE        100
#define DEBUG   1     /* geral */
#define DEBUG_TK 1    /* do ficheiro interf_tk.c */
#define DEBUG_K  1    /* do ficheiro kontrol_a.c */
#define DEBUG_TEL 1   /* do ficheiro telem_a.c */
#define DEBUG_FICH 1  /* da funcao le_ficheiro (misc.c)*/
#define DEBUG_MES 1   /* do ficheiro mensagem.c */
#define DEBUG_SPACE 1 /* do ficheiro space_mouse.c */
#define DEBUG_IMAGEM 1 /* do ficheiro s_img.c */


#endif
```

ficheiro cam.h

```
#include <math.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
/*
 * Camera Declarations
 * ~~~~~~~~~~~~~~~~~~~
 */
void camOpen( int w, int h );
void camClose( void );
void camGrabber( void (*vision)(char *frame) );


/*
 * X11 Declarations
 * ~~~~~~~~~~~~~~~~
 */

struct Win {
  int w, h;
  Display *dsp;
  Window win;
  XImage *img;
  GC gc;
  char *frame;
};
struct Win *winOpen( int w, int h );
void winClose( struct Win *win );
void winShow( struct Win *win, unchar *buffer );
void winShowC( struct Win *win, unchar *buffer );
```

ficheiro comunica.h

```
#ifndef COMUNICA_H_
#define COMUNICA_H_
```

```
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <unistd.h>
#include <stdlib.h>


#define ID_PORTO        5

#define ID_SEND_VAR     50
#define ID_SEND_VAR_ACK 51
#define INIT_TELEM      54
#define START_TELEM     55
#define STOP_TELEM      56

#define START_GAME      70
#define STOP_GAME       71

#define ID_SEND_TELEM_SS  60
#define ID_SEND_TELEM_CS  61

#define ID_PING        10
#define ID_PING_OK     11
#define ID_PING_KO     12
#define ID_PING_64     13
#define ID_PING_256    14
#define ID_PING_1024   15

#define ID_PEDIDO_LIG      1
#define ID_PEDIDO_LIG_ACK  2
#define ID_PEDIDO_LIG_NACK 3
#define ID_ORDEM           100
#define ID_ORDEM_ACK       101
#define ID_ORDEM_NACK      102
#define ID_CTR             110
#define ID_CTR_ACK         111
#define ID_CTR_NACK        112
#define ID_RESET           120
#define ID_VEL_RODA        130

#define ID_SENSIBILITY     140

#define ID_START_GAME      25
#define ID_STOP_GAME       26

#define INIC_COM  0   /* servidor -> cliente
                         (ola', estas vivo?) */
#define INIC1_COM  6   /* servidor -> cliente
                          (ola', estas vivo?) */
#define INFO      1   /* cliente  -> servidor
                         (estou vivo e tenho estes parametros...) */
#define SYNC_IMG  2   /* servidor -> cliente
                         (manda-me tramas de imagem) */
#define IMG       3   /* cliente  -> servidor
                         (aqui vai uma trama com uma fracao de imagem) */
#define SUSP_IMG  4   /* servidor -> cliente
                         (vou suspender a comunicacao, nao mandes nada
                         ate' eu dizer qq coisa) */
#define SUSP1_IMG  5

#define N_PING      3

#define PORTO       20020
#define PORTO1      20021
#define PORTO2      20022
#define PORTO3      20023


#define MAX_BUFFER  500 /* buffer para mandar para o socket (maximo) */
#define MAX_LIG     10
#define TIME_OUT    10   /* segundos ate' dar um erro de time-out numa ligacao */
```

```
#define MAX_NOME    20
#define PORTO_IMG_CLI 20510
#define PORTO_IMG_SER 20520
#define BUF_SIZE 500
#define TAM_LINHA 100

enum ordem {frente, tras, esquerda, direita, para, roda_esq, roda_dir, ctr};

/* estrutura que contem os incrementos
   para a sensibilidade */
struct incr_msg{
 short inc_v,inc_w;
};

struct porto_msg{
 short porto1,porto2;
};

struct ctrl_msg{
   long acc,kp,kd,kl,ki;
};

struct ordem_vel_roda{
   short ordem;
};

struct linha_str{
   char linha[TAM_LINHA];
};



/* parametros a passar do cliente de imagem para o servidor */
struct par_cam
{ int W,H,DIV_N,DIM,DIM_DIV_N,N_SOCK,BYTES_PIXEL;
  char nome[MAX_NOME];
  int id;
  struct par_cam *prox;
};

struct msg
{ short id;               /* identificador da mensagem */
  long tamanho_corpo;     /* tamanho em bytes do corpo da mensagem */
  void* corpo;
};

/*  define um elemento de uma lista de robots */
struct rob_el
{ char* nome;
  int id;
};

int lineariza(void* buff,struct msg *m,int tam);
int deslineariza(struct msg *m,void* buff);

#endif
```

ficheiro global_s.h

```
#ifndef GLOBAL_S
#define GLOBAL_S

#define MAX_VARIAVEIS 20
#define TAMANHO_NOME 80
#define MAX_ROBOTS 10
#define BUFFILE 80
#define N_OPERACOES  6
#define BLOCK        0
#define NO_DESC      0
#define BUF_TELEM_STR 120
```

```
#define MAX_LIGACOES 9


struct controlador_str{
  int kp,kd,ki,kl;
};

struct robot_elem{
  int id;
  char* nome;
  int accel;
  struct controlador_str *cont;
  int plataforma;
};


struct Telem_Str{
   short id;
   short var_id;
   short control;
   short value;
};

struct Telem2_Str{
   short id;
   short var_id;
   char value_str[BUF_TELEM_STR];
};

struct Kontrol_Str{
   short id;
   char nome_var[TAMANHO_NOME];
   short time;
};


struct RobotVar_Str{

  char *nome_robot;
  char *variaveis[MAX_VARIAVEIS];
  int *tempo[MAX_VARIAVEIS];
  int socket_id;
};



/*  protos */


     /* telem */

        int trata_ordem_show(char *,int );
        void serv_show();
        void executa();

     /* kontrol */

        void  Inicializa_Vars();
        void Acrescenta_R(char *);
        void Acrescenta_Var(int ,char *, int );
        int Devolve_Id(char *);
       int Devolve_SockId(char *nome_m);
        void Apaga_Id(int );
       void Apaga_Tudo();
        int le_ficheiro_var(char *);
        void dump_structure();
        int Inicializa_Socket (char *nome_serv);
       void Inicializa_Com();
        int escreve(int id);
/*        void kontrol_main(); */


     /* all */
       /* code include in telem_s.c */
```

```
     void sair1();
     int which(fd_set *);


#endif
```

ficheiro graph.h

```
#include <stdio.h>
#include <stdlib.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <time.h>

#define MAX_GRAPH    20


struct graph_str{

  Display *display;
  Window window;
  int screen;
  unsigned long foreground, background;
  GC   mygc;
  XEvent xevent;
  Font font;
  XSizeHints myhint;
};


/* protos */
void Inicializa_Graph(void);
void  Acrescenta_Graph(int ,char **,char [],int);
void Apaga_Graph(int );
void Apaga_Graph_All();
void Desenha_Grelha(int ,char [],int);
void Escreve_Valor(int ,int ,int );
void Escreve_String(int ,int ,char * );
void Escreve_Var(int ,int ,char *);
```

ficheiro interf_tk.h

```
#include <tcl.h>
#include <tk.h>



int Esq_Fre_Vel_Cmd(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]);

int Fre_Vel_Cmd(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]);

int Dir_Fre_Vel_Cmd(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]);

int Esq_Vel_Cmd(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]);

int Dir_Vel_Cmd(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]);

int Esq_Tra_Vel_Cmd(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]);
```

```
int Tra_Vel_Cmd(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]);

int Dir_Tra_Vel_Cmd(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]);

int Rot_Esq_Vel_Cmd(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]);

int Rot_Dir_Vel_Cmd(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]);

int Aplica_Accel_Cmd(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]);

int Aplica_Cont_Cmd(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]);

int Aplica_Sens_Cmd(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]);

int Reset_Cmd(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]);

int Parar_Cmd(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]);

int Inicializa_Valores(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]);

int Actualizar_Menu(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]);

int Quit_Command(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]);

int Tcl_AppInit ( Tcl_Interp *interp );

int mostra_imagem(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]);

int inic_man(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]);


int Ready_Game_Cmd(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]);

int Ready_Cmd(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]);

int Start_Game_Cmd(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]);


int Stop_Game_Cmd(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]);

int Act_Fich_Telemetria(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]);


int Suspende_Imagem_Cmd(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]);
```

ficheiro lista.h

```
#ifndef lista_h
#define lista_h
struct lista {
```

```
  void* info;
  struct lista *proximo;
};
struct lista *inicializa_lista();

int vazia_lista(struct lista *l);

struct lista *insere_lista(struct lista *l,void *d);

void* devolve_lista(struct lista *l,void *d,int(*comparador)(void*,void*));

struct lista *retira_lista(struct lista *l,void *d,int(*comparador)(void*,void*));

void* devolve_l(struct lista *l);


void limpa_lista(struct lista *l);

struct lista *ordena_lista(struct lista *l,int(*comparador)(void*,void*,void*));

#endif
```

ficheiro mensagem.h

```
#ifndef MENSAGEM_H_
#define MENSAGEM_H_



#include "global_a.h"


/* mensagem protos */

int Actualiza_Cont(struct robot_elem *);
int Ping_Command(struct robot_elem *);
int ordem_roda(int,int );
int ordem_reset(int );
int Ready_Game_Command(int ,char *);
int Ready_Command();
int Start_Game_Command();
int Stop_Game_Command();
int Sens_Command(int,int, int);




#endif
```

ficheiro misc.h

```
#ifndef MISC_H_
#define MISC_H_

#include "lista.h"
#include "c_img.h"
#include "comunica.h"
#include "interface.h"

/* estrutura de parametros  */
struct Param {
  struct Param *next;
  char name[BUF_SIZE];
  int value;
  char *string;
  void *pointer;
  int isNet;
  clock_t times;
};
```

```
/* miscelaneous protos */

int comp(void *,void *);
int comp2(void *,void *,void *);
void imprime_lista(struct lista *);
int grava_end(char *);
int le_ficheiro(char *);
void trata_driver();
int Retira_Activo(char *);
void act_lig(int );
void Actualiza_Ligacoes(int t,struct memoria *);
int le_porto();
int escreve_porto();


/* protos de manipulacao de parametros */
void parmOpen( char *);
void parmDump( void );
int parmGet( char *, int  );


#endif
```

ficheiro space_m.h

```
#ifndef MAGELLAN_H_
#define MAGELLAN_H_

/* magellan protos */

void magellan (void);



#endif
```

Ficheiros de código

ficheiro interfarce.tk

```
#!/usr/local/bin/wish8.0

wm title . ISOCROB
wm geometry . 640x480+20+20
wm minsize . 640 480

menu .menubar -background  darkblue -fg white -relief sunken -borderwidth 3 -
activebackground darkgreen -activeforeground white

. config -bg black  -menu .menubar
. config -bd 2 -relief sunken



######################
# variaveis globais #
######################

set i [menu .menubar.m0]
set f [menu .menubar.m1]
set r [menu .menubar.m2]
set a [menu .menubar.m3]
```

```
set j [menu .menubar.m4]
set h [menu .menubar.m5]

set ficheiro_robots ender
set ficheiro_robots_img ender_img

set fich_man_esc ""
set fich_tel_esc ""

# ficheiro por defeito, esta variavel nao e' suposto mudar nunca
set fich_man_def "parm.dat"
set fich_tel_def "telemetria.dat"

set lista_agentes {}
set lista_agentes_id {}


set var_jog(0) GoalKeeper
set var_goal_aux Blue

for {set inde 1} {$inde < 10} {incr inde} {
   set var_jog($inde) Attacker
}

for {set inde 0} {$inde < 10} {incr inde} {
  set roda_accel($inde) 0
  set incr_var($inde) 50
  set cont_var($inde) 0
  set accel_var($inde) 0
  set var_telem($inde) 1
  set var_active($inde) 1
  set var_goal($inde) Blue
}


######################
##  codigo principal #
######################

.menubar add cascade -label File -underline 0 -menu .menubar.m1
.menubar add cascade -label "Manual Control" -underline 0 -menu .menubar.m2
.menubar add cascade -label "Image" -underline 0 -menu .menubar.m0
.menubar add cascade -label "Update Communications" -underline 0  -menu  .menubar.m3
.menubar add cascade -label Game -underline 0  -menu  .menubar.m4
.menubar add cascade -label About  -underline 0 -menu .menubar.m5

$f add command -label Manual -underline 0 -command {chamar_fich {.} 1} -accelerator
<C>
$f add command -label Default -underline 0 -command {chamar_fich {.} 3 } -accelerator
<D>
$f add separator
$f add command -label Quit -underline 0 -command  quit -accelerator <S>

$j add command -label Telemetry -underline 0 -command {chamar_fich {.} 2} -accelerator
<T>
$j add command -label Default -underline 0 -command {chamar_fich {.} 4 } -accelerator
<D>
$j add command -label "See CATALOG" -underline 0 -command { jan_catalog } -accelerator
<S>

$j add separator

$j add command -label Play -underline 0 -command {jogo var_jog } -accelerator <J>

$a add command -label "Update" -underline 0 -command Actualizar -accelerator <U>
$h add command -label "About Application" -underline 0 -command Creditos -accelerator
<A>
$h add command -label "About Team " -underline 0 -command Team_Info -accelerator <T>


bind . <Key-H> Credits
bind . <Key-O> Open
bind . <Key-A> Go
bind . <Key-L> Vel
```

```
bind . <Key-P> Par
bind . <Key-R> run
bind . <Key-J> game


proc imag {} {

 image create photo socrob1 -file [ file join imagens/socrob1.gif ]
 frame .d -width 250 -height 200 -bg grey80 -relief groove -bd 5
 pack .d -side top
 label .d.do  -bd 3 -image socrob1
 pack .d.do -side top

}







###############
# procedures  #
###############



#########################
# actualiza o menu do robots activos para o controlo
# e imagem
#########################
proc Act_Menu_Tk { } {
        global ficheiro_robots
         global ficheiro_robots_img
        global r
        global i
         $r delete 0 end
         $i delete 0 end
         set id [open $ficheiro_robots r]
         set id2 [open $ficheiro_robots_img r]
         set flag_disp 1
         set flag_disp2 1
         set int 0
         global lista_agentes
         global lista_agentes_id

         set lista_agentes [list {}]
         set lista_agentes_id [list {}]

         while {![eof $id]} {
                 global titulo
                 global titulo2
                 global idrob
                 global desc
                 global idrob2
                 global desc2

                 gets $id titulo
                 gets $id2 titulo2
                 set idrob [ string range $titulo 0  0 ]
                 set desc [ string range $titulo 3  end ]
                 set desc2 [ string range $titulo2 3 end ]

                 if { [string compare $titulo ----------] == 0 } {
                      set flag_disp 0
                 }
                 if { [string compare $titulo2 ----------] == 0 } {
                      set flag_disp2 0
                 }

          if {$flag_disp==1} {
              $r insert end command -label "$titulo" -command "trata_robot $idrob
$desc"
```

```
            lappend lista_agentes $desc
            lappend lista_agentes_id $idrob
         } else {
                if { [string compare $titulo ----------] == 0 } {
                   $r add separator
                } else {
                          $r insert end command -label "$titulo" -state disabled -
command quit
                       }
                }

         if {$flag_disp2==1} {
            $i insert end command -label "$titulo2" -command "mostra_imagem $desc2"
         } else {
               if { [string compare $titulo2 ----------] == 0 } {
                   $i add separator
                } else {
                          $i insert end command -label "$titulo2" -state disabled -
command quit
                      }
                }
          set int [expr $int+1]
        }

        close $id
        close $id2

       $i add separator
       $i add command -label "Turn Off Imagem" -underline 0 -command Desliga_Img_Cmd

}

#########################################################
# janela de wait enquanto as comunicacoes actualizam
#########################################################
proc jan_aguarde { } {
    toplevel .aguarde
    wm geometry  .aguarde +200+200
    label .aguarde.l -text "Update communications... please wait..." \
       -padx 20 -pady 20 -font { times 16 bold }
    label .aguarde.i -bitmap hourglass -padx 10 -pady 20
    pack .aguarde.l .aguarde.i -side right

    tkwait visibility .aguarde
    tkwait visibility .aguarde.l
    tkwait visibility .aguarde.i
}

proc textdump {fi key value index} {
  if {$key == "text" } {
    puts -nonewline $fi "$value"
  }
}

###############
## funcao de debug
###############

proc text_dump {fi t {start 1.0} {end end}} {
  $t dump -command {textdump $fi} $start $end
}


###################################
# funcao de gravacao de um ficheiro
###################################
proc grava_fiche {nome} {
  set fi [open $nome w]

  text_dump $fi .code.text
  close $fi

  toplevel .gravado
```

```
  wm title .gravado Quit
  wm geometry .gravado +250+250

  frame .gravado.frase
  label .gravado.frase.t -text "File saved." -height 2
  label .gravado.frase.b -bitmap info
  pack .gravado.frase.b .gravado.frase.t -side left -padx 5
  pack .gravado.frase
  frame .gravado.bot
  button .gravado.bot.sim -text "Ok" -width 12 \
         -command {destroy .gravado ; destroy .code }
  pack .gravado.bot.sim -side left
  pack .gravado.bot -side top
}
################################
##  grava um ficheiro default  #
################################

proc guarda_fich {nome def} {
  global fich_man_def

  if {[string compare $nome $def]==0} {
    toplevel .atencao
    wm title .atencao ATENCAO
    wm geometry .atencao +225+225

    frame .atencao.frase
    label .atencao.frase.t -text "Do you realy want to change\n \
                                  default settings?"
    label .atencao.frase.b -bitmap question -bg darkred -fg yellow
    pack .atencao.frase.b .atencao.frase.t -side left -padx 5
    pack .atencao.frase
    frame .atencao.bot
    button .atencao.bot.sim -text "Yes" -width 12 -foreground darkred \
           -activebackground darkred -activeforeground yellow \
           -command "destroy .atencao ; grava_fiche $nome"
    button .atencao.bot.nao -text "Opsss... no!" -width 12 \
           -command { destroy .atencao }
    pack .atencao.bot.nao .atencao.bot.sim -side left
    pack .atencao.bot -side top
  } else {
     grava_fiche "$nome"
  }
}



###########################################
#  mostra a janela de escolha de ficheiro #
###########################################

proc jan_fich {nome def} {


  #  puts "os ficheiros: \n$nome\n$def"
  destroy .code

  set id [open [file join $nome]]
  if {![string compare $id ""]==0} {
    toplevel .code
    frame .code.frame
    pack .code.frame -expand yes -fill both -padx 1 -pady 1
    text .code.text -height 30 -width 80 -wrap word\
      -xscrollcommand ".code.xscroll set" \
      -yscrollcommand ".code.yscroll set" \
      -setgrid 1 -highlightthickness 0 -pady 2 -padx 3
    scrollbar .code.xscroll -command ".code.text xview" \
      -highlightthickness 0 -orient horizontal
    scrollbar .code.yscroll -command ".code.text yview" \
      -highlightthickness 0 -orient vertical
    grid .code.text -in .code.frame -padx 1 -pady 1 \
      -row 0 -column 0 -rowspan 1 -columnspan 1 -sticky news
    grid .code.yscroll -in .code.frame -padx 1 -pady 1 \
      -row 0 -column 1 -rowspan 1 -columnspan 1 -sticky news
```

```
    wm title .code "[file join $nome]"
    wm iconname .code $nome
    .code.text delete 1.0 end
    .code.text insert 1.0 [read $id]
    .code.text mark set insert 1.0
    close $id
    button .code.sair -width 12 -text Exit  -command { destroy .code }
    button .code.gravar -width 12 -text Save \
           -command "guarda_fich $nome $def"
    pack .code.gravar .code.sair -side left -expand y
  }
}


#################################################################
#  mostra a janela de visualizacao do catalogo de variaveis #
#################################################################
proc jan_catalog { } {

  puts "janela CATALOG"
  destroy .catalog
  set fich "CATALOG"

  set id [open [file join $fich]]
  if {![string compare $id ""]==0} {
    toplevel .catalog
    frame .catalog.frame
    pack .catalog.frame -expand yes -fill both -padx 1 -pady 1
    text .catalog.text -height 30 -width 80 -wrap word\
      -xscrollcommand ".catalog.xscroll set" \
      -yscrollcommand ".catalog.yscroll set" \
      -setgrid 1 -highlightthickness 0 -pady 2 -padx 3
    scrollbar .catalog.xscroll -command ".catalog.text xview" \
      -highlightthickness 0 -orient horizontal
    scrollbar .catalog.yscroll -command ".catalog.text yview" \
      -highlightthickness 0 -orient vertical
    grid .catalog.text -in .catalog.frame -padx 1 -pady 1 \
      -row 0 -column 0 -rowspan 1 -columnspan 1 -sticky news
    grid .catalog.yscroll -in .catalog.frame -padx 1 -pady 1 \
      -row 0 -column 1 -rowspan 1 -columnspan 1 -sticky news
    wm title .catalog "[file join $fich]"
    wm iconname .catalog $fich
    .catalog.text delete 1.0 end
    .catalog.text insert 1.0 [read $id]
    .catalog.text mark set insert 1.0
    close $id
    button .catalog.sair -width 12 -text Exit  -command { destroy .catalog }

    pack .catalog.sair  -expand y
  }
}



########################################
## funcoes auxiliares para os ficheiros #
########################################
proc proc_fich {w int} {
  set tipos {
        {"configuration files" {.dat}}
        {"all files" *}
        {"text files" {.txt .doc} }
    }

  global fich_man_esc
  global fich_tel_esc
  global fich_man_def
  global fich_tel_def

  set fich_esc_tmp [tk_getOpenFile -filetypes $tipos -parent $w]
  if {!($fich_esc_tmp=="")} {
    .abrir.entr.entr delete 0 end
    .abrir.entr.entr insert 0 $fich_esc_tmp
    .abrir.entr.entr xview end
```

```
    if {$int==1} {
      set $fich_man_esc $fich_esc_tmp
    }
    if {$int==2} {
      set $fich_tel_esc $fich_esc_tmp
    }
  }
}


proc chamar_fich {w int} {

 # ficheiro de comando manual

    global fich_man_esc
    global fich_tel_esc
    global fich_man_def
    global fich_tel_def

  if {$int==1} {
    #global fich_man_esc
    #global fich_def_m
    #global fich_man_def
    if {!($fich_man_esc=="")} {
      set fich_def_m $fich_man_esc
    } else {
        set fich_man_esc $fich_man_def
      }
    toplevel .abrir
    wm geometry  .abrir +200+200
    frame .abrir.boto -relief flat -borderwidth 10
    frame .abrir.entr -relief flat -borderwidth 10
    entry .abrir.entr.entr -textvariable fich_man_esc -width 30
    .abrir.entr.entr xview end
    button .abrir.entr.proc -text Search... \
           -command "proc_fich $w 1" -width 12
    pack .abrir.entr.entr .abrir.entr.proc -side left
    pack .abrir.entr
    button .abrir.boto.sair -text Cancel \
           -command "destroy .abrir .code" -width 12
    button .abrir.boto.ver -text "Edit file" -width 12 \
           -command {jan_fich $fich_man_esc $fich_man_def }
    button .abrir.boto.aceitar -text Accept -width 12 \
           -command { jan_aguarde;
                      inic_man "$fich_man_esc"; \
                      destroy .aguarde .code .abrir }
    pack .abrir.boto.sair .abrir.boto.ver .abrir.boto.aceitar -side left
    pack .abrir.boto

    tkwait visibility .abrir
  }

 # ficheiro de telemetria
  if {$int==2} {
    global fich_tel_esc
    # global fich_def_t
    global fich_tel_def
    if {!($fich_tel_esc=="")} {
      set fich_tel_def $fich_tel_esc
    } else {
        set fich_tel_esc $fich_tel_def
      }
    toplevel .abrir
    wm geometry  .abrir +200+200
    frame .abrir.boto -relief flat -borderwidth 10
    frame .abrir.entr -relief flat -borderwidth 10
    entry .abrir.entr.entr -textvariable fich_tel_esc -width 30
    .abrir.entr.entr xview end
    button .abrir.entr.proc -text Search... \
           -command "proc_fich $w 2" -width 12
    pack .abrir.entr.entr .abrir.entr.proc -side left
    pack .abrir.entr
    button .abrir.boto.sair -text Cancel \
           -command "destroy .abrir .code" -width 12
```

```
    button .abrir.boto.ver -text "Edit file" -width 12 \
           -command {jan_fich $fich_tel_esc $fich_tel_def }
    button .abrir.boto.aceitar -text Accept -width 12 \
           -command {
                     Act_Fich_Telemetria $fich_tel_esc
                     destroy .aguarde .code .abrir }
    pack .abrir.boto.sair .abrir.boto.ver .abrir.boto.aceitar -side left
    pack .abrir.boto

    tkwait visibility .abrir
  }

  if {$int==3} {
    #global fich_man_esc
    #global fich_def_m
    # global fich_man_def
    jan_fich $fich_man_def $fich_man_def
  }

  if {$int==4} {
    #global fich_tel_esc
    # global fich_def_t
    #global fich_tel_def
    jan_fich $fich_tel_def $fich_tel_def
  }

}



proc trata_robot {id desc} {

  controlo_rodas $id $desc

}

proc Actualizar { } {

jan_aguarde
Actualizar_Menu
Act_Menu_Tk

destroy .aguarde

}


################################
# funcao auxiliar para o jogo #
################################

proc conf_bot {w id} {
  $w.botoes.botao$id configure -state disabled
  if {$id==1 } {
    $w.botoes.botao2 configure -state normal
    $w.botoes.botao3 configure -state disabled
    $w.botao_s configure -state disabled
  }
  if {$id==2} {
    $w.botoes.botao1 configure -state disabled
    $w.botoes.botao3 configure -state normal
  }
  if {$id==3} {
    $w.botoes.botao1 configure -state normal
    $w.botoes.botao2 configure -state disabled
    $w.botao_s configure -state normal
  }
}

##############
#  Jogo      #
##############
```

```
proc jogo {var_jog} {

  global array_nome
  global array_id
  global var_goal
  global var_goal_aux

  toplevel .jog
  focus .jog
  grab .jog

  wm title .jog Game
  wm geometry .jog +200+200

  set JO .jog
  set prim 0
  frame $JO.frame -bd 5 -relief groove
  frame $JO.frame1 -bd 5 -relief groove

  frame $JO.frame.label
  label $JO.frame.label.l1 -text Robot -height 2 -width 10
  # label $JO.frame.label.l2 -text Active -height 2 -width 10
  label $JO.frame.label.l3 -text Position -height 2 -width 10
  label $JO.frame.label.l4 -text "Team (GOAL)" -height 2 -width 15
  # label $JO.frame.label.l5 -text "Telemetry" -height 2 -width 10
  pack $JO.frame.label.l1  $JO.frame.label.l3 $JO.frame.label.l4  -side left -padx 25
-pady 5
  pack $JO.frame.label

  # $JO.frame.label.l2
  # $JO.frame.label.l5

  global lista_agentes
  global lista_agentes_id

  set int 0
  foreach index "$lista_agentes" {
    if {$index!=""} {
      set array_nome($int) $index
      set int [expr $int+1]
    }
  }
  set int 0
  foreach index "$lista_agentes_id" {
    if {$index!=""} {
      set array_id($int) $index
      set int [expr $int+1]
    }
  }

  foreach index [array names array_nome] {
            frame $JO.frame.$index
       label $JO.frame.$index.t -text $array_nome($index) -height 2

        # checkbutton $JO.frame.$index.active -text "Active" -variable
var_active($index) -relief flat

       if {$prim == 0} {
       tk_optionMenu $JO.frame.$index.position  var_jog($index) GoalKeeper Attacker
Defender "Penalty Shooter"
              incr prim
       } else {
       tk_optionMenu $JO.frame.$index.position  var_jog($index)  Attacker  GoalKeeper
Defender "Penalty Shooter"
        }

       tk_optionMenu $JO.frame.$index.team  var_goal($index) Blue Yellow

        # checkbutton $JO.frame.$index.telem -text "Telemetry" -variable
var_telem($index) -relief flat

        # $JO.frame.$index.t

        # $JO.frame.$index.telem
```

```
       pack  $JO.frame.$index.t $JO.frame.$index.position $JO.frame.$index.team   -
side left -padx 25 -pady 5
        pack $JO.frame.$index -side top -pady 2
  }



  #frame $JO.frame1.active
  #label $JO.frame1.active.t -text "Active" -height 2
  #checkbutton $JO.frame1.active.aux -text "Active(ALL)" -variable var_active_aux -
relief flat

  #  button $JO.frame1.active.botao -width 8 -text "Apply" -command {
  #    for { set i 0 } {$i < [array size array_nome]} {incr i} {
  #            set var_active($i) $var_active_aux
  #        }
  #  }


  frame $JO.frame1.goal -width 50
  label $JO.frame1.goal.t -text "TEAM GOAL (ALL)" -height 2
  tk_optionMenu $JO.frame1.goal.team  var_goal_aux  Blue Yellow
  button $JO.frame1.goal.botao -width 8 -text "Apply" -command {
       for { set i 0 } {$i < [array size array_nome]} {incr i} {
              set var_goal($i) $var_goal_aux
       }
  }

  #frame $JO.frame1.telem
  #label $JO.frame1.telem.l -text "Telemetry" -height 2
  #checkbutton $JO.frame1.telem.aux -text "Telemetry(ALL)" -variable var_telem_aux -
relief flat
  #button $JO.frame1.telem.botao -width 8 -text "Apply" -command {
  #    for { set i 0 } {$i < [array size array_nome]} {incr i} {
  #            set var_telem($i) $var_telem_aux
  #        }
  #  }


  #pack $JO.frame1.active.t $JO.frame1.active.aux $JO.frame1.active.botao -side left -
padx 25 -pady 5
  #pack $JO.frame1.active -side top -pady 2


  pack $JO.frame1.goal.t $JO.frame1.goal.team $JO.frame1.goal.botao -side left -padx
25 -pady 5
  pack $JO.frame1.goal -side top -pady 2

  #pack $JO.frame1.telem.l $JO.frame1.telem.aux $JO.frame1.telem.botao -side left -
padx 25 -pady 5
  #pack $JO.frame1.telem -side top -pady 2

  pack $JO.frame -pady 10 -padx 10
  pack $JO.frame1 -pady 10 -padx 10

  frame $JO.botoes


  button $JO.botoes.botao1 -width 8 -text "Ready Game" -command {
     conf_bot .jog 1
     Suspende_Imagem_Cmd;
     foreach index [array names array_nome] {
         set linha($index) ""
         switch -exact $var_jog($index) {
            Attacker { set linha($index) [append linha($index) " -m1"] }
           GoalKeeper { set linha($index) [append linha($index) " -m0"] }
         }
        if { $var_goal($index)=="Blue" } {
            set linha($index) [append linha($index) " -g1"]
         } else {
            set linha($index) [append linha($index) " -g2"]
         }
         set linha($index) [append linha($index) " -k -p -n"]
```

```
    };

    foreach index1 [array names array_nome] {
      puts -nonewline "agente: '"
      puts -nonewline $array_nome($index1)
      puts -nonewline "'      id: '"
      puts -nonewline $array_id($index1)
      puts -nonewline "'      linha de comando: '"
      puts -nonewline $linha($index1)
      puts "'"
      Ready_Game_Cmd $array_id($index1) $linha($index1)
    };

    Ready_Cmd;

  }

  button $JO.botoes.botao2 -width 8 -text "Start Game" \
     -command "conf_bot .jog 2; Start_Game_Cmd "
  button $JO.botoes.botao4 -width 8 -text "Ready Game" \
     -command "conf_bot .jog 4; Ready_Game_Cmd "
  button $JO.botoes.botao3 -width 8 -text "End   Game" \
     -command "conf_bot .jog 3; Stop_Game_Cmd "

  button $JO.botao_s -width 8 -text "Quit" -command { destroy .jog }

  conf_bot .jog 3

  pack $JO.botoes.botao1 $JO.botoes.botao2 $JO.botoes.botao3 -side left -fill x -ipadx
20 -padx 10

  pack $JO.botoes
  set a [array size array_nome]


  pack $JO.botao_s -side top -pady 5
  tkwait window .jog

}

###################
#  quit          #
###################

proc quit {} {
  toplevel .sair
  wm title .sair   Quit
  wm geometry  .sair +200+200

  frame .sair.frase
  label .sair.frase.t -text "Do you realy want to leave" -height 2
  label .sair.frase.b -bitmap question  -bg darkred -fg yellow
  pack .sair.frase.b .sair.frase.t -side left -padx 5
  pack .sair.frase
  frame .sair.bot
  button .sair.bot.sim -text "Yes!" -width 12 -foreground darkred \
         -activebackground darkred -activeforeground yellow \
         -command { Quit_Command ; destroy . }
  button .sair.bot.nao -text "Opsss... no!" -width 12 -command { destroy .sair }
  pack .sair.bot.nao .sair.bot.sim -side left
  pack .sair.bot -side top
}


###################
#  Creditos      #
###################

proc Creditos {} {

 toplevel .creditos

 wm title .creditos "About Aplicattion..."
```

```
 wm geometry .creditos +200+200
 frame .creditos.quadro1
 frame .creditos.quadro1.f
 set texto .creditos.quadro1.f

 label $texto.1 -text "TFC 98/99" -height 3 -font  { times 22 bold }
 label $texto.2 -text "Created by:" -height 2
 label $texto.3 -text "Bruno Miguel Correia Nº41332" -height 2
 label $texto.4 -text "Sergio Saraiva Nº41440" -height 2
 label $texto.branca -text " " -height 2

 image create photo lrmlogo -file [ file join imagens/logo.gif ]
 image create photo isrlogo -file [ file join imagens/isrlogo.gif ]
 image create photo istlogo -file [ file join imagens/istlogo.gif ]
 image create photo logo -file [ file join    imagens/logo.gif ]
 image create photo linuxlogo -file [ file join imagens/linux.gif ]

 frame .creditos.quadro2
 button .creditos.quadro2.botao -text "      OK      " -command { destroy .creditos }
-padx 2m

 frame .creditos.quadro1.imagem
 set imagem .creditos.quadro1.imagem
 label $imagem.imagem1 -image lrmlogo -bd 2 -relief flat
 label $imagem.imagem2 -image isrlogo -bd 2 -relief sunken
 label $imagem.imagem3 -image istlogo -bd 2 -relief sunken
 label .creditos.quadro2.img -image linuxlogo -bd 2 -relief flat


 pack $imagem.imagem1 $imagem.imagem2 $imagem.imagem3 -side left -anchor e
 pack $texto.1
 pack $texto.2 $texto.3 $texto.4 -side top -anchor w
 pack $texto $imagem -side left -padx 2m
 pack .creditos.quadro2.img .creditos.quadro2.botao -side left -padx 20
 pack .creditos.quadro1 .creditos.quadro2 -side top

 focus .creditos
 grab .creditos
 tkwait window .creditos
}


###################
#  Team Info     #
###################

proc Team_Info {} {

 toplevel .team

 wm title .team "About Team..."
 frame .team.quadro1
 frame .team.quadro1.f
 set texto .team.quadro1.f

 frame .team.quadro -relief sunken
 label .team.quadro.texto -text "Team  98/99" -height 3 -font  { times 22 bold } -padx
2 -pady 2
 pack .team.quadro.texto


 label $texto.2 -text  "Composed by:"
 label $texto.3 -text  " Professors: "
 label $texto.4 -text  "         Pedro Lima "
 label $texto.5 -text  "         Luis Custodio "
 label $texto.6 -text  "         Carlos Pinto Ferreira "
 label $texto.7 -text  " Msc. Students:"
 label $texto.8 -text  "         Rodrigo Ventura"
 label $texto.9 -text  "         Carlos Marques "
 label $texto.10 -text "         Pedro Aparicio "
 label $texto.11 -text " UnderGraduate Students:"
 label $texto.12 -text "         Bruno Correia  "
 label $texto.13 -text "         Sergio Saraiva "
 label $texto.14 -text "         Filipe Neves   "
```

```
 label $texto.15 -text "         Ricardo Silva  "
 label $texto.16 -text "         Rodrigo Almeida"
 label $texto.branca -text " "

 image create photo isrlogo -file [ file join imagens/isrlogo.gif ]
 image create photo istlogo -file [ file join imagens/istlogo.gif ]
 image create photo logo -file [ file join    imagens/logo1.gif ]
 image create photo linuxlogo -file [ file join imagens/linux.gif ]

 frame .team.quadro2
 button .team.quadro2.botao -text "      OK      " -command { destroy .team } -padx 2m



 label .team.quadro1.imagem  -image logo -bd 2 -relief flat

 frame .team.quadro.img

 label .team.quadro.img.imagem2 -image isrlogo -bd 2 -relief sunken
 label .team.quadro.img.imagem3 -image istlogo -bd 2 -relief sunken

 pack .team.quadro.img.imagem2 .team.quadro.img.imagem3 -side left -anchor e
 pack .team.quadro.texto .team.quadro.img  -side left -expand y -padx 40


 pack  $texto.2 $texto.branca  $texto.3  $texto.4 $texto.5 $texto.6 $texto.7 $texto.8
$texto.9 $texto.10 $texto.11 $texto.12 $texto.13 $texto.14 $texto.15 $texto.16  -side
top -anchor w
 pack $texto .team.quadro1.imagem -side left -padx 2m


 label .team.quadro2.img -image linuxlogo -bd 2 -relief flat
 pack .team.quadro2.img .team.quadro2.botao -side left -padx 20
 pack .team.quadro .team.quadro1 .team.quadro2 -side top -pady 2m

 focus .team
 grab .team
 tkwait window .team
}



#########################################################
#  funcoes auxiliares para a funcao controlo_rodas  #
#########################################################

proc conf_sens {id valor} {
  global incr_var
  set incr_var($id) $valor
}

proc sens_aplica {id} {
  global incr_var
  puts $incr_var($id)
  puts $id
  Aplica_Sens_Cmd $id $incr_var($id)
}

proc sens_processa { w var} {
  global incr_var
  global x
  if {$x==1} {
    set x 0;
    pack forget $w.incr $w.control $w.parte6
    pack forget $w.incr.botao_check
    pack forget $w.incr.botao_aplica
    pack $w.incr.botao_check $w.incr.scl \
    $w.incr.botao_aplica -side top -expand 1 -padx 2
#    $w.incr.scl set $incr_var($var)
    actualiza_janela $w
  } else {
      set x 1;
      pack forget $w.incr.scl $w.incr.botao_aplica
  }
```

```
}

proc accel_processa { w var } {
  global accel_var
  if {$accel_var($var)==1} {
     pack forget $w.aceleracao $w.control $w.parte6
     pack forget $w.aceleracao.botao_check
     pack forget $w.aceleracao.botao_aplica
     pack $w.aceleracao.botao_check $w.aceleracao.accel_var \
     $w.aceleracao.botao_aplica -side left -expand 1 -padx 2
     actualiza_janela $w
   } else {
       pack forget $w.aceleracao.botao_aplica $w.aceleracao.accel_var
   }
}

proc accel_aplica { desc var } {
  global roda_accel

  puts $desc
  puts $var

 Aplica_Accel $desc $roda_accel($var)
}

proc cont_aplica { desc var } {
  global var_kp
  global var_kd
  global var_ki
  global var_kl

  puts $desc
  puts $var

  Aplica_Cont $desc $var_kp($var) $var_kd($var) $var_ki($var) $var_kl($var)
}


proc cont_processa { w var } {
  global cont_var
  puts "dentro do cont_processa"
  puts $cont_var($var)
  if { $cont_var($var)==1 }  {
           puts ife
           subst {puts $w }
           pack forget $w.parte6
            pack $w.control.cont_button  $w.control.fields $w.control.botao -side top -
expand y -pady 2
            actualiza_janela  $w
      } else {
                puts else
                 pack $w
                pack forget $w.control.fields
                 pack forget $w.control.botao
      }
}

proc actualiza_janela {j} {
   puts $j
   pack forget $j.cabecalho $j.cont_tipo2 $j.incr $j.aceleracao $j.control $j.parte6
   pack $j.cabecalho $j.cont_tipo2 $j.incr $j.aceleracao $j.control $j.parte6 -side
top -padx 10 -pady 4
   pack $j
}


proc Gravar {id desc} {
  set tipos {
        {"configuration files" {.dat}}
        {"all files" *}
        {"text files" {.txt .doc} }
    }
```

```
  set fich_grav [tk_getSaveFile -filetypes $tipos -parent {.} -initialfile
sem_titulo.dat -defaultextension .dat]

  if {![string compare $fich_grav ""]==0} {
    global var_kp
    global var_ki
    global var_kd
    global var_kl
    global cont_var
    global roda_accel

    set fim [string length $desc]
    set fim [expr $fim-2]
    set str [string range $desc 0 $fim]
    set param ""
    append param "accel $roda_accel($id)\n"
    append param "kp $var_kp($id)\n"
    append param "kd $var_kd($id)\n"
    append param "ki $var_ki($id)\n"
    append param "kl $var_kl($id)\n"

    actualiza_ficheiro "$fich_grav" $id $desc $str $param
  }
}


##########################################
# funcao que efectua o controlo das rodas #
##########################################

proc controlo_rodas {id desc} {

  global incr_var
  set j rodas$id
  toplevel .$j

  wm title .$j $desc
  wm geometry .$j +200+200

  frame .$j.controlo
  set w .$j.controlo

  frame $w.cabecalho

  image create photo img1 -file [file join imagens/logo.gif]
  label $w.cabecalho.imagem -image img1 -bd 1 -relief flat -anchor nw
  label $w.cabecalho.titulo -text "Wheels control" -relief ridge  \
     -bd 1 -borderwidth 2 -width 20 -font  { times 16 bold }

  pack $w.cabecalho.titulo  $w.cabecalho.imagem  -side left -padx 2



  image create photo esqfre -file [file join imagens/esqfre.gif]
  image create photo fre    -file [file join imagens/fre.gif]
  image create photo dirfre -file [file join imagens/dirfre.gif]
  image create photo esq  -file [file join imagens/esq.gif]
  image create photo para -file [file join imagens/para.gif]
  image create photo dir  -file [file join imagens/dir.gif]
  image create photo esqtra -file [file join imagens/esqtra.gif]
  image create photo tra    -file [file join imagens/tra.gif]
  image create photo dirtra -file [file join imagens/dirtra.gif]
  image create photo rodesq -file [file join imagens/rodesq.gif]
  image create photo roddir -file [file join imagens/roddir.gif]

  frame $w.cont_tipo2
  frame $w.cont_tipo2.linha1
  frame $w.cont_tipo2.linha2
  frame $w.cont_tipo2.linha3
  frame $w.cont_tipo2.linha4

  button $w.cont_tipo2.linha1.esqfre -image esqfre -command "Esq_Fre_Vel $id"
  button $w.cont_tipo2.linha1.fre    -image fre -command "Fre_Vel $id"
  button $w.cont_tipo2.linha1.dirfre -image dirfre -command "Dir_Fre_Vel $id"
```

```
  button $w.cont_tipo2.linha2.esq  -image esq -command "Esq_Vel $id"
  button $w.cont_tipo2.linha2.para -image para -command "Parar $id"
  button $w.cont_tipo2.linha2.dir  -image dir -command "Dir_Vel $id"
  button $w.cont_tipo2.linha3.esqtra -image esqtra -command "Esq_Tra_Vel $id"
  button $w.cont_tipo2.linha3.tra    -image tra -command "Tra_Vel $id"
  button $w.cont_tipo2.linha3.dirtra -image dirtra -command "Dir_Tra_Vel $id"
  button $w.cont_tipo2.linha4.rodesq -image rodesq -command "Rot_Esq_Vel $id"
  button $w.cont_tipo2.linha4.roddir -image roddir -command "Rot_Dir_Vel $id"


  pack $w.cont_tipo2.linha1.esqfre $w.cont_tipo2.linha1.fre
$w.cont_tipo2.linha1.dirfre -side left
  pack $w.cont_tipo2.linha2.esq $w.cont_tipo2.linha2.para $w.cont_tipo2.linha2.dir -
side left
  pack $w.cont_tipo2.linha3.esqtra $w.cont_tipo2.linha3.tra
$w.cont_tipo2.linha3.dirtra -side left
  pack $w.cont_tipo2.linha4.rodesq $w.cont_tipo2.linha4.roddir -side left
  pack $w.cont_tipo2.linha1 $w.cont_tipo2.linha2 $w.cont_tipo2.linha3
$w.cont_tipo2.linha4 \
    -side top


       ###################
       #  Sensibilidade  #
       ###################

  frame $w.incr -borderwidth 1 -relief raised
  global x
  set x 1
  checkbutton $w.incr.botao_check -text "Sensibility (%)" \
              -relief flat -width 15 -anchor w  \
              -command "sens_processa $w $id" \
               -offvalue 0 -onvalue 1
  pack $w.incr.botao_check -expand 1

  scale $w.incr.scl -orient horizontal -length 250 -from 0 -to 100 \
       -tickinterval 10 -showvalue true -command "conf_sens $id"
  $w.incr.scl set $incr_var($id)
  button $w.incr.botao_aplica -width 8 -relief flat -text "Apply" \
         -command "sens_aplica $id "

       ################
       #  Aceleracao  #
       ################

  frame $w.aceleracao -borderwidth 1 -relief raised
  entry $w.aceleracao.accel_var -width 10 -relief sunken -textvariable
"roda_accel($id)"
  button $w.aceleracao.botao_aplica -width 8 -relief flat -text "Apply" -command "
accel_aplica $desc $id "
  checkbutton $w.aceleracao.botao_check  -text "Acceleration" -variable accel_var($id)
\
              -relief flat  -offvalue 0 -onvalue 1 -width 15 -anchor w\
              -command "accel_processa $w $id"

  pack $w.aceleracao.botao_check -expand 1

       ###############
       # Controlador #
       ###############

   frame $w.control -borderwidth 1 -relief raised
   frame $w.control.fields

   label $w.control.fields.label_kp -text "KP:"
   label $w.control.fields.label_ki -text "KI:"
   label $w.control.fields.label_kd -text "KD:"
   label $w.control.fields.label_kl -text "KL:"
   entry $w.control.fields.entry_kp -width 5 -relief sunken -textvariable
"var_kp($id)"
   entry $w.control.fields.entry_ki -width 5 -relief sunken -textvariable
"var_ki($id)"
   entry $w.control.fields.entry_kd -width 5 -relief sunken -textvariable
"var_kd($id)"
```

```
   entry $w.control.fields.entry_kl -width 5 -relief sunken -textvariable
"var_kl($id)"
   button $w.control.botao -width 8 -text "Apply" -command " cont_aplica $desc $id "

   pack $w.control.fields.label_kp $w.control.fields.entry_kp \
         $w.control.fields.label_ki $w.control.fields.entry_ki \
         $w.control.fields.label_kd $w.control.fields.entry_kd \
         $w.control.fields.label_kl $w.control.fields.entry_kl \
         -side left -padx 2 -pady 2;

   checkbutton $w.control.cont_button  -text "Controlers" -variable cont_var($id)
-relief flat -width 15 -anchor w -onvalue 1 -offvalue 0\
               -command "cont_processa $w $id"

   pack $w.control.cont_button

        ##################
       # Botoes finais #
       ##################

  frame $w.parte6 -relief flat

  frame $w.parte6.p1 -borderwidth 12
  button $w.parte6.p1.botao1 -width 8 -text "Stop" -command " Parar $id"
  button $w.parte6.p1.botao2 -width 8 -text "Reset" -command " Reset $id"
  frame $w.parte6.p2 -borderwidth 1 -relief groove
  button $w.parte6.p2.botao4 -width 8 -text "Save" -command " Gravar $id $desc"
  button $w.parte6.p2.botao3 -width 8 -text "Quit" -command " destroy .$j "
  pack $w.parte6.p1.botao1 $w.parte6.p1.botao2 -side left -fill x -expand y -padx 2m -
pady 2
  pack $w.parte6.p2.botao4 $w.parte6.p2.botao3 -side left -fill x -expand y -padx 2m -
pady 2

  pack $w.parte6.p1 $w.parte6.p2 -side top

  pack $w.cabecalho $w.cont_tipo2 $w.incr $w.aceleracao  $w.control  $w.parte6 -side
top  -padx 10 -pady 4
  pack $w

}
#fim do controlo_rodas




## executa as funcoes de broadcast e
## inicializa os valores

imag
Act_Menu_Tk
Inicializa_Valores



#fim de ficheiro
```

ficheiro interface.c

```
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>
#include "comunica.h"
#include "lista.h"
#include "c_img.h"
#include "cam.h"
#include "sem.h"
#include "misc.h"
#include "space_mouse.h"
#include "mensagem.h"
```

```
#include "interface.h"
#include "interf_tk.h"
#include "global_s.h"


/* variaveis globais usadas em toda a aplicacao */
struct lista *agentes_desactivo,*agentes_activo;
int Socket_Lookup[MAX_LIGACOES];
int magellan_current=-1;
int sock_telemetria;
int PORT,PORT2;
char buffer[MAX_BUFFER];
struct memoria partilha;
struct RobotVar_Str* Vars[MAX_ROBOTS];
char *fich_telemetria;
pthread_t thread_telemetria,thread_kontrol;


void agente_SpaceMouse()
{
   if (DEBUG_SPACE) printf("Sou o micro agente SpaceMouse \n");
     magellan();
}

void agente_Interface()
{  char *argv[2];

   argv[0]=malloc(sizeof(char)*20);
   argv[1]=malloc(sizeof(char)*20);
   strcpy(argv[0],"interface");
   strcpy(argv[1],"interface.tk");


   Tk_Main(2,argv,Tcl_AppInit);
}


void main( int argc,char *argv[] )
{
  pthread_t space_mouse_pth,interface_pth,rede_mem_pth,mem_ecra_pth;
  int i;
  /* ficheiro que contem os valores default para a tele-comando*/
  char *fich="parm.dat";

  /* define o ficheiro de telemetria */
  fich_telemetria=(char *)calloc(80,sizeof(char));
  strcpy(fich_telemetria,"telemetria.dat");

  le_porto();

  /* intercepcao dos signals */
  signal(SIGINT,sair);
  signal(SIGKILL,sair);
  signal(SIGQUIT,sair);
  signal(SIGPIPE,SIG_IGN);
  signal(SIGALRM,nada);
  fclose(NULL);

  sync(); sync(); sync();

  /* inicializacao da tabela de sockets */
  for (i=0;i<MAX_LIGACOES;i++) Socket_Lookup[i]=-1;
  agentes_desactivo=inicializa_lista();
  agentes_activo=inicializa_lista();

  /* inicializacao dos semaforos para sincronizar o acesso aos buffers */
  inisem(NMAX,SEMMUTEX,SEMALOC,&partilha.mutex1,&partilha.semaloc1);
  inisem(NMAX,SEMMUTEX2,SEMALOC2,&partilha.mutex2,&partilha.semaloc2);
  partilha.activo=NULL;
  partilha.pendente=NULL;
  partilha.sair=0;
  partilha.parado=0;
  partilha.par=NULL;
```

```
  /* ficheiro que contem os enderecos dos robots, aos quais se pretende
     obter a imagem */
  parmOpen("enderecos");

  if(!le_ficheiro(fich))
   printf("ERRO: leitura do ficheiro : %s \n",fich);

  /* actualiza as ligacoes dos robots activos*/
  Actualiza_Ligacoes(1,&partilha);

  if(!grava_end("ender"))
    printf("ERRO: na gravacao do ficheiro ender \n");

  /* cria a tarefa do space_mouse */
 if(pthread_create(&space_mouse_pth,NULL,(void*)agente_SpaceMouse,NULL))
   { printf("ERRO: a criar a tarefa space_mouse \n");
     sair();
   }
  /* cria a tarefa da interface */
 if(pthread_create(&interface_pth,NULL,(void*)agente_Interface,NULL))
   { printf("ERRO: a criar a tarefa interface \n");
     sair();
   }
  /* cria as tarefas relativas a imagem */
  if(pthread_create(&rede_mem_pth,NULL,(void*)rede_mem,(void*)&partilha))
  { printf("ERRO: a criar a tarefa imagem (rede_mem_pth)\n");
    sair();
  }
  if(pthread_create(&mem_ecra_pth,NULL,(void*)mem_ecra,(void*)&partilha))
  { printf("ERRO: a criar a tarefa imagem(mem_ecra_pth)\n");
    sair();
  }


  pthread_join(space_mouse_pth,NULL);
  pthread_join(interface_pth,NULL);
  pthread_join(rede_mem_pth,NULL);
  pthread_join(mem_ecra_pth,NULL);

  free(partilha.ptr1);
  free(partilha.ptr2);

  sair(); /* sai */
}
```

ficheiro comunica.c

```
#include "comunica.h"

int lineariza(void* buff,struct msg *m,int tam)
{ int s1,s2;
  s1=sizeof(m->id);
  s2=sizeof(m->tamanho_corpo);

  m->id=htons(m->id);
  m->tamanho_corpo=htonl(tam);
  memcpy(buff,&m->id,s1);
  memcpy(buff+s1,&m->tamanho_corpo,s2);
  memcpy(buff+s1+s2,m->corpo,tam);
  return s1+s2+tam;
}

int deslineariza(struct msg *m,void* buff)
{ int s1,s2;
  s1=sizeof(m->id);
  s2=sizeof(m->tamanho_corpo);

  memcpy(&m->id,buff,s1);
  memcpy(&m->tamanho_corpo,buff+s1,s2);
  m->id=ntohs(m->id);
  m->tamanho_corpo=ntohl(m->tamanho_corpo);
```

```
  memcpy(m->corpo,buff+s1+s2,m->tamanho_corpo);
  return s1+s2+m->tamanho_corpo;
}
```

ficheiro cam.c

```
/*
 * Includes
 * ~~~~~~~~
 */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <signal.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <signal.h>
#include <sys/times.h>
#include <sys/mman.h>
#include <time.h>
#include <getopt.h>
#include <unistd.h>

#include <linux/types.h>
#include <linux/videodev.h>
#include <asm/spinlock.h>
#include <pthread.h>

#include "cam.h"
#include "interface.h"

/*
 * Global Data
 * ~~~~~~~~~~~
 */
int USEX = 1;
int USEM = 1;

int camera;
int kbd;

char *cam_map;
struct video_mmap cam_conf = { 0, 200, 100, VIDEO_PALETTE_RGB555 };

/*
 * X11 Functions
 * ~~~~~~~~~~~~~
 */

struct Win *winOpen( int w, int h ) {
  struct Win *win;
  XVisualInfo vis;

  static int geometryOffsetX = 10;
  static int geometryOffsetY = 10;

  if ( USEX ) {
    sync();
    win = malloc( sizeof(struct Win) );
    win->w = w;
    win->h = h;
    win->dsp = XOpenDisplay( getenv("DISPLAY") );
    if ( win->dsp ) {
      if (DEBUG_IMAGEM) printf("X11 ok -- using it\n");
      win->win = XCreateSimpleWindow( win->dsp, DefaultRootWindow(win->dsp),
                                      geometryOffsetX, geometryOffsetY, w, h,
```

```
                                    0, 0, 0 );
      geometryOffsetX += w + 30;
      XMapWindow( win->dsp, win->win );
      XMatchVisualInfo( win->dsp, DefaultScreen(win->dsp), 16, TrueColor, &vis );
      win->gc = XCreateGC( win->dsp, win->win, 0, NULL );
      win->frame = malloc( 2*w*h );
      win->img = XCreateImage( win->dsp, vis.visual, 16, ZPixmap, 0,
                               win->frame, w, h, 16, 2*w );
      XInitImage( win->img );
      return win;
    } else {
      if (DEBUG_IMAGEM) printf("Can't open display, switching off X11\n");
      free(win);
      USEX = 0;
      return NULL;
    }
  } else
    return NULL;
}



void winClose( struct Win *win ) {
  if ( USEX ) {
    XDestroyWindow( win->dsp, win->win );
    XCloseDisplay( win->dsp );
    free( win );
  }
}


/* B&W version */
void winShow( struct Win *win, unchar *buffer ) {
  int i,j,y;

  if ( USEX ) {
    for ( i=0 ; i<(win->w)*(win->h) ; i++ ) {
      y = buffer[i];
      j = i<<1;
      (win->img->data)[j]   = (y>>3 & 0x1f) | (y<<3 & 0xe0);
      (win->img->data)[j+1] = (y>>5 & 0x07) | (y & 0xf8);
    }
    XPutImage( win->dsp, win->win, win->gc, win->img, 0, 0, 0, 0,
               win->w, win->h );
  }
}


/* Color version */
void winShowC( struct Win *win, unchar *buffer ) {
  int i,r,g,b;

  if ( USEX ) {
    for ( i=0 ; i<2*(win->w)*(win->h) ; i+=2 ) {
      r = (buffer[i+1] & 0x7c) >> 2;
      g = ((buffer[i+1] & 0x03) << 3) | ((buffer[i] & 0xe0) >> 5);
      b = buffer[i] & 0x1f;

      (win->img->data)[i]   = (b & 0x1f) | (g<<6 & 0xc0);
      (win->img->data)[i+1] = (g>>2 & 0x07) | (r<<3 & 0xf8);
    }
    XPutImage( win->dsp, win->win, win->gc, win->img, 0, 0, 0, 0,
               win->w, win->h );
  }
}
```

ficheiro graph.c

```
#include <stdio.h>
#include <stdlib.h>
#include <X11/Xlib.h>
```

```
#include <X11/Xutil.h>
#include <time.h>
#include <math.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/times.h>
#include <time.h>
#include <getopt.h>
#include <readline/readline.h>
#include <signal.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>

#include "graph.h"


struct graph_str* graph[MAX_GRAPH];


/* inicializa a estrutura graph */
void Inicializa_Graph(void){

  int i;

  for(i=0;i<MAX_GRAPH;i++)
     graph[i]=NULL;

}

void  Acrescenta_Graph(int argc,char *argv[],char name[],int num_var){

  int i;

  for(i=0;graph[i];i++);
  graph[i]=(struct graph_str*)malloc(sizeof(struct graph_str));

  /* connect to the X server */

  graph[i]->display = XOpenDisplay ("");

  if (graph[i]->display == NULL) {
       fprintf (stderr, "cannot connect to server\n");
       exit (EXIT_FAILURE);
  }

  /* get default screen */
  graph[i]->screen = DefaultScreen(graph[i]->display);

   /* get black and white representation on current screen */
  graph[i]->background = BlackPixel(graph[i]->display, graph[i]->screen);
  graph[i]->foreground = WhitePixel(graph[i]->display, graph[i]->screen);

  graph[i]->myhint.x=200; graph[i]->myhint.y=300;
  graph[i]->myhint.width=800;graph[i]->myhint.height=45+20*num_var;
  graph[i]->myhint.flags=PPosition|PSize;

  graph[i]->window = XCreateSimpleWindow (graph[i]->display,
  DefaultRootWindow(graph[i]->display), graph[i]->myhint.x, graph[i]->myhint.y,
graph[i]->myhint.width,graph[i]->myhint.height, 2,
  graph[i]->foreground, graph[i]->background);

  XSetStandardProperties(graph[i]->display,graph[i]-
>window,name,name,None,argv,argc,&(graph[i]->myhint));


  if (graph[i]->window == 0) {
         fprintf (stderr, "cannot open window\n");
         exit(EXIT_FAILURE);
  }

  XSelectInput(graph[i]->display, graph[i]->window, ExposureMask);
```

```
  /* pop this window up on the screen */
  XMapRaised(graph[i]->display, graph[i]->window);

  graph[i]->mygc=XCreateGC(graph[i]->display,graph[i]->window, 0,0);
  XSetBackground(graph[i]->display,graph[i]->mygc,graph[i]->background);
  XSetForeground(graph[i]->display,graph[i]->mygc,graph[i]->foreground);

  XNextEvent (graph[i]->display, &(graph[i]->xevent));
   /* flush X request queue to server */

  XFlush(graph[i]->display);

}


void Apaga_Graph(int i){

  if (graph[i]){
    XFreeGC(graph[i]->display,graph[i]->mygc);
    XDestroyWindow(graph[i]->display,graph[i]->window);
    XCloseDisplay(graph[i]->display);
    free(graph[i]);
    graph[i]=NULL;
  }

}

void Apaga_Graph_All(){

 int i;
 for(i=0;i<MAX_GRAPH;i++) Apaga_Graph(i);

 }



void Desenha_Grelha(int id,char nome_robot[],int num_var){

 int i;
 char var[30]="TELEMETRY   ,   MACHINE :"   ;


 XDrawString(graph[id]->display,graph[id]->window,graph[id]-
>mygc,70,15,var,strlen(var));
 XDrawString(graph[id]->display,graph[id]->window,graph[id]-
>mygc,230,15,nome_robot,strlen(nome_robot));

 XDrawString(graph[id]->display,graph[id]->window,graph[id]->mygc,5,35,"Variable
Name",strlen("Variable Name"));
 XDrawString(graph[id]->display,graph[id]->window,graph[id]-
>mygc,235,35,"Value",strlen("Value"));
 /*
 XDrawString(graph[id]->display,graph[id]->window,graph[id]-
>mygc,355,35,"Other",strlen("Other"));
 */

 for (i=0;i<num_var;i++){
    XDrawRectangle(graph[id]->display,graph[id]->window,graph[id]-
>mygc,5,40+i*20,780,20);
 }

 XDrawLine(graph[id]->display,graph[id]->window,graph[id]-
>mygc,230,40,230,40+20*num_var);
 /*
 XDrawLine(graph[id]->display,graph[id]->window,graph[id]-
>mygc,350,40,350,40+20*num_var);
 */

 XFlush(graph[id]->display);
}


void Escreve_Valor(int id,int num_var,int valor){
```

```
  char var[20];


  sprintf(var,"%d ",valor);

  /*    limpa a area antiga  */
    XSetForeground(graph[id]->display,graph[id]->mygc,graph[id]->background);
    XFillRectangle(graph[id]->display,graph[id]->window,graph[id]-
>mygc,240,45+num_var*20,100,15);
    XSetForeground(graph[id]->display,graph[id]->mygc,graph[id]->foreground);


  XDrawString(graph[id]->display,graph[id]->window,graph[id]-
>mygc,240,55+num_var*20,var,strlen(var));
  XFlush(graph[id]->display);

}

void Escreve_String(int id,int num_var,char *string){

  char var[150];


  strcpy(var,string);
  strcat(var,"\0");

  /*    limpa a area antiga  */
    XSetForeground(graph[id]->display,graph[id]->mygc,graph[id]->background);
    XFillRectangle(graph[id]->display,graph[id]->window,graph[id]-
>mygc,240,45+num_var*20,540,15);
    XSetForeground(graph[id]->display,graph[id]->mygc,graph[id]->foreground);

  XDrawString(graph[id]->display,graph[id]->window,graph[id]-
>mygc,240,55+num_var*20,var,strlen(var));


  XFlush(graph[id]->display);

}



void Escreve_Var(int id,int num_var,char *nome){

 char n[50];

 strcpy(n,nome);
 XDrawString(graph[id]->display,graph[id]->window,graph[id]-
>mygc,15,55+num_var*20,n,strlen(n));
}
```

ficheiro inter_tk.c

```
#include <tcl.h>

#include <tk.h>
#include "interf_tk.h"
#include "mensagem.h"
#include "comunica.h"
#include "lista.h"
#include "misc.h"
#include "c_img.h"
#include "interface.h"

extern struct lista *agentes_desactivo;
extern struct lista  *agentes_activo;

extern struct memoria partilha;

extern char *fich_telemetria;
```

```
/*************************/
/* Controlo das rodas    */
/*************************/

int Esq_Fre_Vel_Cmd(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]) {

  if (DEBUG_TK)  printf("\n O Butao Esquerda-Frente pressionado\n");
  ordem_roda(esquerda,atoi(argv[1]));
  ordem_roda(frente,atoi(argv[1]));
  return TCL_OK;}

int Fre_Vel_Cmd(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]) {

  if (DEBUG_TK)  printf("\n O Butao Frente pressionado\n");
  ordem_roda(frente,atoi(argv[1]));
  return TCL_OK;
}

int Dir_Fre_Vel_Cmd(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]) {

  if (DEBUG_TK)  printf("\n O Butao Direita-Frente pressionado\n");
  ordem_roda(direita,atoi(argv[1]));
  ordem_roda(frente,atoi(argv[1]));


  return TCL_OK;
}

int Esq_Vel_Cmd(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]) {

  ordem_roda(esquerda,atoi(argv[1]));

  if (DEBUG_TK)  printf("\n O Butao Esquerda pressionado\n");
  return TCL_OK;
}

int Dir_Vel_Cmd(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]) {

  ordem_roda(direita,atoi(argv[1]));

  if (DEBUG_TK)  printf("\n Butao Direita pressionado\n");
  return TCL_OK;
}

int Esq_Tra_Vel_Cmd(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]) {


  ordem_roda(esquerda,atoi(argv[1]));
  ordem_roda(tras,atoi(argv[1]));

  if (DEBUG_TK)  printf("\n Butao Esquerda-Tras pressionado\n");
  return TCL_OK;
}

int Tra_Vel_Cmd(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]) {


  ordem_roda(tras,atoi(argv[1]));

  if (DEBUG_TK)  printf("\n Butao Tras pressionado\n");
  return TCL_OK;
}

int Dir_Tra_Vel_Cmd(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]) {
```

```
  ordem_roda(direita,atoi(argv[1]));
  ordem_roda(tras,atoi(argv[1]));

  if (DEBUG_TK)  printf("\n O Butao Direita-Tras pressionado\n");
  return TCL_OK;
}


int Rot_Esq_Vel_Cmd(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]) {

  ordem_roda(roda_esq,atoi(argv[1]));


  if (DEBUG_TK)  printf("\n Butao Roda-Esquerda pressionado\n");
  return TCL_OK;
}

int Rot_Dir_Vel_Cmd(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]) {

  ordem_roda(roda_dir,atoi(argv[1]));
  if (DEBUG_TK)  printf("\n Butao Roda-Direita pressionado\n");
  return TCL_OK;
}

int Aplica_Accel_Cmd(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]) {

  int Accel;
  struct robot_elem *el;

  if (DEBUG_TK) printf("\n  %s %s \n",argv[1],argv[2]);
  Accel=atoi(argv[2]);


  if (DEBUG_TK)  printf("\n Butao Aceleracao Pressionado");
  if (DEBUG_TK)  printf(" \n RE: acelaracao da roda %d \n",Accel);

  el=devolve_lista(agentes_activo,argv[1],comp);
  el->accel=Accel;
  Actualiza_Cont(el);

  return TCL_OK;
}


int Aplica_Cont_Cmd(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]) {
int kp,kd,ki,kl;
struct robot_elem *el;

  kp=atoi(argv[2]);
  kd=atoi(argv[3]);
  ki=atoi(argv[4]);
  kl=atoi(argv[5]);

  if (DEBUG_TK) printf("\n argv[0]  -> %s \n",argv[0]);
  if (DEBUG_TK) printf("\n Butao Aplica Controlador Pressionado");
  if (DEBUG_TK) printf("\n valores dos controladores ");
  if (DEBUG_TK) printf("\n kp -> %d,   kd -> %d   ki -> %d  kl ->%d \n",kp,kd,ki,kl);


  el=devolve_lista(agentes_activo,argv[1],comp);
  el->cont->kp=kp;el->cont->kd=kd;el->cont->ki=ki;el->cont->kl=kl;
  Actualiza_Cont(el);

  return TCL_OK;
}


int Aplica_Sens_Cmd(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]) {
```

```
   int valor;
   int id;
   id=atoi(argv[1]);
   valor=atoi(argv[2]);
   if (DEBUG_TK) printf("\n valor ->  %d \n",valor);

   valor=valor*100;
   if (DEBUG_TK)  printf("\n valor ->  %d \n",valor);
   Sens_Command(id,valor,valor);


return TCL_OK;
}



int Reset_Cmd(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]) {

 ordem_reset(atoi(argv[1]));

 if (DEBUG_TK) printf(" \n Butao Reset pressionado \n");

 return TCL_OK;
}


int Parar_Cmd(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]) {

  ordem_roda(para,atoi(argv[1]));
  if(DEBUG_TK)   printf(" \n O Butao Parar foi pressionado \n");

  return TCL_OK;

}


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

int Inicializa_Valores(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]) {

  struct robot_elem *el;
  char *var,*t;


  struct lista *aux;


   var=(char *)malloc(10*sizeof(char));
   t=(char *)malloc(10*sizeof(char));
  aux=agentes_activo;

  while (aux){
    el=(struct robot_elem *)devolve_l(aux);

    if (DEBUG_TK) printf("accel %d \n",el->accel);
    if (DEBUG_TK) printf(" nome %s \n",el->nome);
    sprintf(t,"%d",el->accel);
    sprintf(var,"roda_accel(%d)",el->id);
    if (DEBUG_TK) printf(" var %s value %s \n",var,t);
    Tcl_SetVar(interp,var,t,0);

    if (!el->plataforma) {
      sprintf(t,"%d",el->cont->kp);
      sprintf(var,"var_kp(%d)",el->id);
      Tcl_SetVar(interp,var,t,0);

    if (DEBUG_TK)   printf(" var %s value %s \n",var,t);

      sprintf(t,"%d",el->cont->kd);
      sprintf(var,"var_kd(%d)",el->id);
      Tcl_SetVar(interp,var,t,0);
```

```
    if (DEBUG_TK)   printf(" var %s value %s \n",var,t);

      sprintf(t,"%d",el->cont->ki);
      sprintf(var,"var_ki(%d)",el->id);
      Tcl_SetVar(interp,var,t,0);

    if (DEBUG_TK)   printf(" var %s value %s \n",var,t);

      sprintf(t,"%d",el->cont->kl);
      sprintf(var,"var_kl(%d)",el->id);
      Tcl_SetVar(interp,var,t,0);

    if (DEBUG_TK)   printf(" var %s value %s \n",var,t);

    }

    aux=aux->proximo;
   }

return 0;
}


int Actualizar_Menu(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]) {

 Actualiza_Ligacoes(0,&partilha);
 grava_end("ender");
return 0;
}


int Quit_Command(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]) {

 trata_driver();
return 0;
}


int mostra_imagem(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[])
{
  char *nome;
  struct par_cam *par;
  int bingo;
  nome=argv[1];

  for(par=partilha.par,bingo=0;par;par=par->prox)
  { if(!strcmp(par->nome,nome))
    { partilha.pendente=par;
      bingo=!bingo;
      break;
    }
  }
  if(!bingo)
  { if (DEBUG_TK) printf("agente %s nao disponivel\n",nome);
    partilha.parado=1;
  }
  else
  { if (DEBUG_TK) printf("agente %s activo\n",nome);
    partilha.parado=0;
  }
  return 0;
}

int inic_man(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[])
{ if(!le_ficheiro(argv[1]))
     if (DEBUG_TK) printf("ERRO: leitura do ficheiro : %s \n",argv[1]);

  Actualiza_Ligacoes(1,&partilha);
```

```
  if(!grava_end("ender"))
    if (DEBUG_TK) printf("ERRO: na gravacao do ficheiro \n");
  return 0;
}


int Suspende_Imagem_Cmd(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]){
 if (DEBUG_TK) printf("\n dentro do ciclo do suspende \n ");
 Suspende_Imagem();
 sleep(1);

return 0;
}


int Desliga_Img_Cmd(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[])
{
 partilha.parado=1;
 return 0;
}


int Ready_Game_Cmd(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]){

 int id;
 char *l;

 id=atoi(argv[1]);

 l=(char *)calloc(20,sizeof(int));
  strcpy(l,argv[2]);
 /* strcpy(l," -m1 -g1 -k -p -n"); */

 printf("\n string -> %s \n",l);


 Ready_Game_Command(id,l);

 return 0;
}

int Ready_Cmd(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]){

 Ready_Command();

 return 0;
}


int Start_Game_Cmd(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]){


 Start_Game_Command();

 return 0;
}


int Stop_Game_Cmd(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]){


 Stop_Game_Command();
 Activa_Imagem();


 return 0;
}
```

```
int Act_Fich_Telemetria(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[]){

 strcpy(fich_telemetria,argv[1]);
 return 0;
}


int actualiza_ficheiro(ClientData clientData, Tcl_Interp *interp,
                int argc, char *argv[])
{ FILE *fd;
  char str[MAX_BUFFER],aux[MAX_BUFFER];
  int bingo;

  bzero(str,MAX_BUFFER);

  fd=fopen(argv[1],"r+");

  if (DEBUG_TK) printf("\n actualiza_ficheiro \n");
  sprintf(str,"# %s %s %s\n",argv[2],argv[3],argv[4]);

  for(bingo=0;(!feof(fd))||(!bingo);)
  { fgets(aux,MAX_BUFFER,fd);
    if (DEBUG_TK) printf("1. parm: %s",aux);
    if (DEBUG_TK) printf("2. parm: %s",str);
    if(!strcmp(aux,str))
    { fputs(argv[5],fd);
      bingo=1;
    }
  }
  if (!bingo)
  { fputs(str,fd);
    fputs("\n",fd);
    fputs(aux,fd);
  }
  if (DEBUG_TK) printf(" outro %s",argv[5]);

  fclose(fd);
  return 0;
}



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

int Tcl_AppInit ( Tcl_Interp *interp )
{

  if ( Tcl_Init ( interp ) == TCL_ERROR ){
    return TCL_ERROR;
  }

  if ( Tk_Init ( interp ) == TCL_ERROR ){
    return TCL_ERROR;
  }

  Tcl_CreateCommand ( interp, "inic_man", inic_man,
                      (ClientData)NULL, (Tcl_CmdDeleteProc *)NULL );

  Tcl_CreateCommand ( interp, "Act_Fich_Telemetria", Act_Fich_Telemetria,
                      (ClientData)NULL, (Tcl_CmdDeleteProc *)NULL );

  Tcl_CreateCommand ( interp, "mostra_imagem", mostra_imagem,
                      (ClientData)NULL, (Tcl_CmdDeleteProc *)NULL );

  Tcl_CreateCommand ( interp, "Start_Game_Cmd", Start_Game_Cmd,
                      (ClientData)NULL, (Tcl_CmdDeleteProc *)NULL );

  Tcl_CreateCommand ( interp, "Ready_Game_Cmd", Ready_Game_Cmd,
                      (ClientData)NULL, (Tcl_CmdDeleteProc *)NULL );

  Tcl_CreateCommand ( interp, "Ready_Cmd", Ready_Cmd,
                      (ClientData)NULL, (Tcl_CmdDeleteProc *)NULL );
```

```
  Tcl_CreateCommand ( interp, "Stop_Game_Cmd", Stop_Game_Cmd,
                      (ClientData)NULL, (Tcl_CmdDeleteProc *)NULL );

  Tcl_CreateCommand ( interp, "Actualizar_Menu", Actualizar_Menu,
                      (ClientData)NULL, (Tcl_CmdDeleteProc *)NULL );

  Tcl_CreateCommand ( interp, "Quit_Command", Quit_Command,
                      (ClientData)NULL, (Tcl_CmdDeleteProc *)NULL );

  Tcl_CreateCommand ( interp, "Inicializa_Valores", Inicializa_Valores,
                      (ClientData)NULL, (Tcl_CmdDeleteProc *)NULL );

  Tcl_CreateCommand ( interp, "Aplica_Accel", Aplica_Accel_Cmd,
                      (ClientData)NULL, (Tcl_CmdDeleteProc *)NULL );

  Tcl_CreateCommand ( interp, "Aplica_Cont", Aplica_Cont_Cmd,
                      (ClientData)NULL, (Tcl_CmdDeleteProc *)NULL );

  Tcl_CreateCommand ( interp, "Aplica_Sens_Cmd", Aplica_Sens_Cmd,
                      (ClientData)NULL, (Tcl_CmdDeleteProc *)NULL );

  Tcl_CreateCommand ( interp, "Esq_Fre_Vel", Esq_Fre_Vel_Cmd,
                      (ClientData)NULL, (Tcl_CmdDeleteProc *)NULL );

  Tcl_CreateCommand ( interp, "Fre_Vel", Fre_Vel_Cmd,
                      (ClientData)NULL, (Tcl_CmdDeleteProc *)NULL );

  Tcl_CreateCommand ( interp, "Dir_Fre_Vel", Dir_Fre_Vel_Cmd,
                      (ClientData)NULL, (Tcl_CmdDeleteProc *)NULL );

  Tcl_CreateCommand ( interp, "Esq_Vel", Esq_Vel_Cmd,
                      (ClientData)NULL, (Tcl_CmdDeleteProc *)NULL );

  Tcl_CreateCommand ( interp, "Dir_Vel", Dir_Vel_Cmd,
                      (ClientData)NULL, (Tcl_CmdDeleteProc *)NULL );

  Tcl_CreateCommand ( interp, "Esq_Tra_Vel", Esq_Tra_Vel_Cmd,
                      (ClientData)NULL, (Tcl_CmdDeleteProc *)NULL );

  Tcl_CreateCommand ( interp, "Tra_Vel", Tra_Vel_Cmd,
                      (ClientData)NULL, (Tcl_CmdDeleteProc *)NULL );

  Tcl_CreateCommand ( interp, "Dir_Tra_Vel", Dir_Tra_Vel_Cmd,
                      (ClientData)NULL, (Tcl_CmdDeleteProc *)NULL );

  Tcl_CreateCommand ( interp, "Rot_Esq_Vel", Rot_Esq_Vel_Cmd,
                      (ClientData)NULL, (Tcl_CmdDeleteProc *)NULL );

  Tcl_CreateCommand ( interp, "Rot_Dir_Vel", Rot_Dir_Vel_Cmd,
                      (ClientData)NULL, (Tcl_CmdDeleteProc *)NULL );

  Tcl_CreateCommand ( interp, "Reset", Reset_Cmd,
                      (ClientData)NULL, (Tcl_CmdDeleteProc *)NULL );

  Tcl_CreateCommand ( interp, "Parar", Parar_Cmd,
                      (ClientData)NULL, (Tcl_CmdDeleteProc *)NULL );

  Tcl_CreateCommand ( interp, "Suspende_Imagem_Cmd", Suspende_Imagem_Cmd,
                      (ClientData)NULL, (Tcl_CmdDeleteProc *)NULL );

  Tcl_CreateCommand ( interp, "Desliga_Img_Cmd", Desliga_Img_Cmd,
                      (ClientData)NULL, (Tcl_CmdDeleteProc *)NULL );

  Tcl_CreateCommand ( interp, "actualiza_ficheiro", actualiza_ficheiro,
                      (ClientData)NULL, (Tcl_CmdDeleteProc *)NULL );


  return TCL_OK;
}
```

ficheiro kontrol_s.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/times.h>
#include <time.h>
#include <getopt.h>
#include <readline/readline.h>
#include <signal.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <pthread.h>

#include "interface.h"
#include "comunica.h"
#include "global_s.h"
#include "graph.h"
#include "lista.h"

/* variaveis globais utilizadas neste ficheiro */
extern int sock_telemetria;
extern struct lista  *agentes_activo;
extern struct RobotVar_Str* Vars[MAX_ROBOTS];
extern   pthread_t   thread_telemetria;
extern int PORT;



void  Inicializa_Vars() {

 int i;
 for (i=0;i<MAX_ROBOTS;i++) Vars[i]=NULL;
 return ;

}

void Acrescenta_R(char *nome_robot){

  char *nome;
  int i,j;

  /* encontra o primeiro slot livre */
  for (i=0;Vars[i];i++);

  Vars[i]=(struct RobotVar_Str*) malloc(sizeof(struct RobotVar_Str));
  nome=(char *)malloc(TAMANHO_NOME*sizeof(char));
  strcpy(nome,nome_robot);
  Vars[i]->nome_robot=nome;

  /* inicializa vars */

  for(j=0;j<MAX_VARIAVEIS;j++){
        Vars[i]->variaveis[j]=NULL;
        Vars[i]->tempo[j]=NULL;
  }

}

void Acrescenta_Var(int i,char *nome_var, int time) {

 int j;

 for (j=0;Vars[i]->variaveis[j];j++);

 Vars[i]->variaveis[j]=(char *)malloc(TAMANHO_NOME*sizeof(char));
 strcpy(Vars[i]->variaveis[j],strcat(nome_var,"\0"));
 Vars[i]->tempo[j]=(int *)malloc(sizeof(int));
 *Vars[i]->tempo[j]=time;
```

```
}

 /* dado o nome da maquina devolve o seu identificador */
 /* o.w. devolve -1 */

int Devolve_Id(char *nome_m) {

 int i;

 for(i=0;Vars[i];i++) {
   if(!strcmp(nome_m,Vars[i]->nome_robot))
   return i;
 }

 return -1;
}

 /* dado o nome da maquina devolve o seu socket id */
 /* o.w. devolve -1 */

int Devolve_SockId(char *nome_m) {

 int i;

 for(i=0;Vars[i];i++) {
   if(!strcmp(nome_m,Vars[i]->nome_robot))
   return Vars[i]->socket_id;
 }

 return -1;
}


void Apaga_Id(int i){

  int j;

  for(j=0;Vars[i]->variaveis[j];j++){

    free(Vars[i]->variaveis[j]);
    free(Vars[i]->tempo[j]);
  }

   free(Vars[i]->nome_robot);
   Vars[i]=NULL;
}




void Apaga_Tudo(){

  int i;

  for(i=0;Vars[i];i++)
     Apaga_Id(i);

}

int Actualiza_Var(){
 struct lista* aux;
 struct robot_elem *el;

  aux=agentes_activo;
  while(aux){
    el=(struct robot_elem *)devolve_l(aux);
    Acrescenta_R(el->nome);
    aux=aux->proximo;
  }
return 1;
}
```

```
int le_ficheiro_var(char *ficheiro){

 FILE *fd;
 static char line[BUFFILE],str1[BUFFILE],str2[BUFFILE];
 int num,id;

 if (DEBUG_K) printf("A ler o ficheiro %s \n",ficheiro);

 Actualiza_Var();   /* inicializa a estrutura var com a lista de agentes
                       activos */

 fd=fopen(ficheiro,"r");
 if (fd) {
   while(!feof(fd)){

      fgets(line,BUFFILE,fd);
      if (sscanf(line,"%s %s %d ",str1,str2,&num)==3){
           if (DEBUG_K) printf("\n (1) -> %s ",str1);
           if (DEBUG_K) printf(" (2) -> %s ",str2);
           if (DEBUG_K) printf(" (3) -> %d \n",num);

           if (Devolve_Id(str1)!=-1) {
              id=Devolve_Id(str1);
              Acrescenta_Var(id,str2,num);
           }
      } else {
        printf("Erro: A estrutura do ficheiro %s esta incorrecta\n",ficheiro);
        fclose(fd);return(-1);
      }

   }
 }
 fclose(fd);

return 1;
}



/* funcao de debug que mostra a estrutura da variaveis de
   telemetria */
void dump_structure(){

  int i,j;

  for(i=0;Vars[i];i++) {
    printf("\n ------------\n");
    printf("Dump structure \n");
    printf("Name of machine -> %s \n",Vars[i]->nome_robot);
    for(j=0;Vars[i]->variaveis[j];j++){
          printf("    Nome da var -> %s ",Vars[i]->variaveis[j]);
          printf("    tempo -> %d \n",*Vars[i]->tempo[j]);
    }
  }
}


int Inicializa_Socket (char *nome_serv){

  int fd;
  char *chr;
  struct sockaddr_in servidor;
  struct  hostent *hp;

  if (DEBUG_K) printf("\n ----- \n Kontol_A.c \n ");
  chr=calloc(10,sizeof(char));
  fd=socket(AF_INET,SOCK_STREAM,0);
  if(fd<0){
        if (DEBUG_K) printf("\n ERRO: na criacao do socket \n");
       strcpy(chr,"KO\0");
        fd=-1;
  } else
       if(!(hp=gethostbyname(nome_serv))) {
         if (DEBUG_K) printf("ERRO: maquina desconhecida %s\n",nome_serv);
```

```
          strcpy(chr,"KO\0");
          close(fd);
          fd=-1;
        } else{
          bzero((char*)&servidor,sizeof(servidor));
          bcopy(hp->h_addr,(char*)&servidor.sin_addr,hp->h_length);
          servidor.sin_family=AF_INET;
          servidor.sin_port=htons(PORT);
          if(connect(fd,(struct sockaddr*)&servidor,sizeof(servidor))<0)
            { if (DEBUG_K) printf("ERRO: a conectar o socket stream ,maquina %s
\n",nome_serv);
            strcpy(chr,"KO\0");

            fd=-1;
            } else {
                if(DEBUG_K) printf("ligacao com sucesso no ""canal"" [%d]\n",fd);
            }
        }

     if(!strcmp(chr,"KO\0")){
       if (DEBUG_K) printf("\n Erro: Na criacao do socket  (%s)\n",nome_serv);
       return -1;
      }

  if (DEBUG_K) printf("\n -------- \n");
 return fd;
}


void Inicializa_Com(){

 int i;

 for(i=0;Vars[i];i++){
   Vars[i]->socket_id=Inicializa_Socket(Vars[i]->nome_robot);
   printf("\n id -> %d \n",Vars[i]->socket_id);
 }

 printf("\n depois do inicializa_com \n");
}


int escreve(int id)
{
  struct msg m;
  int tam;
  struct Kontrol_Str t;
  char buffer[MAX_BUFFER];
  int i,sock_fd;


  sock_fd=Vars[id]->socket_id;

  for(i=0;Vars[id]->variaveis[i];i++){
    t.id=id;
    strcpy(t.nome_var,"\0");
    strcpy(t.nome_var,Vars[id]->variaveis[i]);
    t.time=*(Vars[id]->tempo[i]);

    t.id=htons(t.id);
    t.time=htons(t.time);

    m.id=ID_SEND_VAR;
    m.corpo=&t;
    tam=lineariza(buffer,&m,sizeof(struct Kontrol_Str));

    printf("\n no escreve \n");
    if(write(sock_fd,buffer,tam)<0) printf("\n Erro: (escreve,Kontrol_a.c)  a escrever
no socket \n");
    if(read(sock_fd,buffer,tam)<0) printf("\n Erro: (escreve,Kontrol_a.c) a ler  no
socket \n");
    deslineariza(&m,buffer);
    if (m.id==ID_SEND_VAR_ACK) if (DEBUG_K) printf("\n SEND_VAR_ACK \n");
  }
```

```
  return 0;
}


int teste(int id){

  int sock_fd;
  struct msg m;
  int tam;
  char buffer[MAX_BUFFER];


  sock_fd=Vars[id]->socket_id;

  getchar();


  m.id=START_GAME;
  m.corpo=NULL;
  tam=lineariza(buffer,&m,0);
  if(write(sock_fd,buffer,tam)<0) printf("\n Erro:  a escrever no socket \n");

  getchar();

  printf("\n init \n");
  m.id=INIT_TELEM;
  m.corpo=NULL;
  tam=lineariza(buffer,&m,0);
  if(write(sock_fd,buffer,tam)<0) printf("\n Erro:  a escrever no socket \n");

  getchar();

  printf("\n stop \n");
  m.id=STOP_GAME;
  m.corpo=NULL;
  tam=lineariza(buffer,&m,0);
     close(sock_telemetria);
  if(write(sock_fd,buffer,tam)<0) printf("\n Erro:  a escrever no socket \n");


return 0;
}
```

ficheiro lista.c

```
#include "lista.h"
#include <stdlib.h>

struct lista *inicializa_lista()
{ return NULL;
}

int vazia_lista(struct lista *l)
{ return l==NULL;
}

struct lista *insere_lista(struct lista *l,void *d)
{ struct lista *aux;
  aux=malloc(sizeof(struct lista));
  aux->proximo=l;
  aux->info=d;
  l=aux;
  return l;
}

void* devolve_lista(struct lista *l,void *d,int(*comparador)(void*,void*))
{ struct lista *aux;
  if(!vazia_lista(l))
    for(aux=l;(aux)&&(!comparador(aux->info,d));aux=aux->proximo);
    if(aux) return aux->info; else return NULL;
}
```

```
void* devolve_l(struct lista *l){
  if(!vazia_lista(l))
    return l->info;
  else
    return NULL;
}

struct lista *retira_lista(struct lista *l,void *d,int(*comparador)(void*,void*))
{ struct lista *aux,*segue,*ret;

  if(l)
    for(aux=segue=l;(aux)&&(!comparador(aux->info,d));
        segue=aux,aux=aux->proximo);
  else return NULL;
  if(aux)
    if(aux==l) ret=segue->proximo;
    else {
      segue->proximo=aux->proximo;
      ret=l;
    }

  return ret;
}

void limpa_lista(struct lista *l){

  struct lista *aux,*segue;

  segue=l;

  while(aux){
    segue=aux->proximo;
    free(aux->info);
    free(aux);
    aux=segue;
  }

return;
}


struct lista *ordena_lista(struct lista *l,int(*comparador)(void*,void*,void*)){

  struct lista *aux,*aux2,*actual,*ordenada;

  int valor,valor1,v,flag;

  ordenada=NULL;
  aux=l;
  valor=20;
  valor1=-1;
  flag=1;

  while(flag){
    flag=0;
    while(aux){

      if ((v=comparador(aux->info,&valor,&valor1))!=-1){
        actual=aux;
        valor1=v;
        flag=1;
      }
      aux=aux->proximo;

    }

    if (flag){
      aux2=malloc(sizeof(struct lista));
      aux2->proximo=ordenada;
      aux2->info=actual->info;
      ordenada=aux2;
    }
    aux=l;
```

```
    valor=valor1;
    valor1=-1;
  }
    aux=l;
    while(aux){
      aux2=aux->proximo;
      free(aux);
      aux=aux2;
    }

return ordenada;
}
```

ficheiro mensagem.c

```
#include <netinet/in.h>
#include <pthread.h>

#include "mensagem.h"
#include "comunica.h"
#include "interface.h"
#include "global_a.h"
#include "lista.h"

extern int Socket_Lookup[MAX_LIGACOES];
extern pthread_t thread_telemetria;
extern pthread_t thread_kontrol;
extern  struct RobotVar_Str *Vars[MAX_ROBOTS];
extern int sock_telemetria;
extern int PORT;
extern int PORT2;
extern char *fich_telemetria;
extern struct lista *agentes_activo;

int existe;

int Actualiza_Cont(struct robot_elem *re){
  struct ctrl_msg cm;
  struct msg m;
  int tam;
  char buffer[MAX_BUFFER];

  if (DEBUG_MES) printf("\n Actualiza_Cont: kp-> %d, kd -> %d, ki -> %d, kl -> %d,
accel-> %d \n",re->cont->kp,re->cont->kd,re->cont->ki,re->cont->kl,re->accel);

  if (!re->plataforma){
    cm.kp=htonl(re->cont->kp);
    cm.kd=htonl(re->cont->kd);
    cm.ki=htonl(re->cont->ki);
    cm.kl=htonl(re->cont->kl);
  } else {
    cm.kp=0;cm.kd=0;cm.ki=0;cm.kl=0;
  }
  cm.acc=htonl(re->accel);
  m.id=ID_CTR;
  m.corpo=&cm;

  tam=lineariza(buffer,&m,sizeof(struct ctrl_msg));

  if(write(Socket_Lookup[re->id],buffer,tam)<0){
    printf("ERRO: (Actualiza_Cont,mensagem.c) a escrever no socket, maquina %s",re-
>nome);
    return 0;
  }

return 1;
}
```

```
int Sens_Command(int id,int incv,int incw){

  struct msg m;
  int tam;
  char buf[MAX_BUFFER];
  struct incr_msg i;
  if (DEBUG_MES) printf("\n estou no sens_command \n");

  if (DEBUG_MES) printf("\n IncW -> %d \n IncV -> %d \n",incv,incw);
  i.inc_v=htons((short)incv);
  i.inc_w=htons((short)incv);

  m.id=ID_SENSIBILITY;
  m.corpo=&i;

  tam=lineariza(buf,&m,sizeof(struct incr_msg));

  if(write(Socket_Lookup[id],buf,tam)<0){
    printf("ERRO:(sens_command,mensagem.c) a escrever no socket, id %d",id);
    return 0;
  }
return 0;
}


int Ping_Command(struct robot_elem *re){
  struct msg m;
  int tam;
  char buf[MAX_BUFFER];


  m.id=ID_PING;
  m.corpo=NULL;

  tam=lineariza(buf,&m,0);

  if(write(Socket_Lookup[re->id],buf,tam)<0){
    printf("ERRO:(ping_command,mensagem.c) a escrever no socket, maquina %s",re-
>nome);
    return 0;
  }
  if(read(Socket_Lookup[re->id],buf,tam)<0){
    printf("ERRO:(ping_command,mensagem.c) a escrever no socket, maquina %s",re-
>nome);
    return 0;
  }

  deslineariza(&m,buf);
  if (m.id==ID_PING_OK) return 1;
  else return 0;
}


int ordem_roda(int  o_r,int n_ag)
{ struct msg m;
  struct ordem_vel_roda ordem;
  int tam;
  char buffer[MAX_BUFFER];

  ordem.ordem=htons((short)o_r);
  m.id=ID_ORDEM;
  m.corpo=&ordem;

  tam=lineariza(buffer,&m,sizeof(struct ordem_vel_roda));

  if (n_ag!=-1)
  if(write(Socket_Lookup[n_ag],buffer,tam)<0) printf("ERRO:(ordem_roda,mensagem.c) a
escrever no socket");

  return 0;
}

int Ready_Game_Command(int n_robot,char *linha_arg) {
```

```
  struct msg m;
  int tam;
  char buffer[MAX_BUFFER];
  struct linha_str l;
  struct porto_msg p;


  if (DEBUG_MES) printf("\n ----------\n Ready_Game_Command \n");

  p.porto1=htons((short)PORT);
  p.porto2=htons((short)PORT2);
  m.corpo=&p;
  m.id=ID_PORTO;
  tam=lineariza(buffer,&m,sizeof(struct porto_msg));
  if(write(Socket_Lookup[n_robot],buffer,tam)<0) printf("erro a escrever no socket");

  strcpy(l.linha,linha_arg);
  m.id=ID_START_GAME;
  m.corpo=&l;
  tam=lineariza(buffer,&m,sizeof(struct linha_str));

  if(write(Socket_Lookup[n_robot],buffer,tam)<0) printf("erro a escrever no socket");




return 1;
}

int Ready_Command() {

  char *nome_f;

  int i,existe;

 sleep(3);
  nome_f=(char *)malloc(TAMANHO_NOME*sizeof(char));
  strcpy(nome_f,fich_telemetria);

  Inicializa_Vars();
  le_ficheiro_var(nome_f);
  Inicializa_Com();

  /*  dump_structure(); */
  usleep(100000);

 existe=0;
  for(i=0;Vars[i];i++)
    if(Vars[i]->socket_id!=-1){
      escreve(i); existe=1;sleep(1);}


  if (existe) pthread_create(&thread_telemetria,NULL,(void *) serv_show, NULL);
  sleep(3);


return 1;
}


int Start_Game_Command(){

  struct msg m;
  int tam,sock_fd,i;
  char buffer[MAX_BUFFER];

  if (DEBUG_MES) printf("----------\n Start_Game_Command\n");

  for(i=0;Vars[i];i++)
    if(Vars[i]->socket_id!=-1){

      sock_fd=Vars[i]->socket_id;
```

```
      m.id=START_GAME;
      m.corpo=NULL;
      tam=lineariza(buffer,&m,0);
      if(write(sock_fd,buffer,tam)<0) printf("\n Erro:  a escrever no socket \n");

      /*  sleep(1);
      m.id=INIT_TELEM;
      m.corpo=NULL;
      tam=lineariza(buffer,&m,0);
      if(write(sock_fd,buffer,tam)<0) printf("\n Erro:  a escrever no socket \n");
      */
    }
return 1;
}


int Stop_Game_Command(){

  struct msg m;
  int tam,sock_fd;
  char buffer[MAX_BUFFER];


  PORT=PORT+2;
  PORT2=PORT2+2;
  sock_fd=Vars[0]->socket_id;


  if (DEBUG_MES) printf("----------\n Stop_Game_Command\n");


  m.id=ID_STOP_GAME;
  m.corpo=NULL;
  tam=lineariza(buffer,&m,0);

  if(write(Socket_Lookup[1],buffer,tam)<0) printf("erro a escrever no socket");
  if(write(Socket_Lookup[3],buffer,tam)<0) printf("erro a escrever no socket");



  /*  m.id=STOP_GAME;
  m.corpo=NULL;
  tam=lineariza(buffer,&m,0);

  if(write(sock_fd,buffer,tam)<0) printf("\n Erro:  a escrever no socket \n");
  */


  /*  close(sock_telemetria); */
  pthread_join(thread_telemetria,NULL);

  fclose(NULL);

/* apaga toda a estrutura Vars */
  Apaga_Tudo();


return 1;
}







int ordem_reset(int n_robot) {

  struct msg m;
  int tam;
  char buffer[MAX_BUFFER];

  m.id=ID_RESET;
```

```
  m.corpo=NULL;

  tam=lineariza(buffer,&m,0);

  if(write(Socket_Lookup[n_robot],buffer,tam)<0) printf("ERRO: (ordem_reset,mesagem.c)
a escrever no socket");

  if (DEBUG_MES) printf("\n ordem_reset executada \n");

return 0;
}
```

ficheiro misc.c

```
#include <stdio.h>
#include <stdlib.h>
#include "misc.h"
#include "mensagem.h"
#include "lista.h"
#include "interface.h"
#include "comunica.h"
extern struct lista *agentes_desactivo;
extern struct lista *agentes_activo;
extern int Socket_Lookup[MAX_LIGACOES];
extern struct memoria partilha;
extern int PORT,PORT2;

/* funcao de comparacao de strings */
int comp(void *r,void *d){
  struct robot_elem *el;
  char *desc;
  int res;

  el=(struct robot_elem *)r;
  desc=(char *)d;
  res= (!(strcmp(el->nome,desc)));

return res;
}

/* funcao de comparacao de strings */
int comp2(void *r,void *d,void *d1){
  struct robot_elem *el;
  int *value;
  int *value1;

  el=(struct robot_elem *)r;
  value=(int *)d;
  value1=(int *)d1;

   if ((el->id<*value)&&(el->id>*value1))
     return el->id;
   else return -1;

}


/* funcao que tem como objectivo imprimir uma lista de agentes. funcao de
   debug */
void imprime_lista(struct lista *l)
{ struct robot_elem *a;
  struct lista *aux;
  for(aux=l;aux;aux=aux->proximo)
  { a=(struct robot_elem *)aux->info;
    printf("\n nome: %s\tid: %d\n",a->nome,a->id);
  }
}
```

```
/* grava a lista de robots activos para um ficheiro */
int grava_end(char *ficheiro){

  FILE *fd;
  struct lista* aux;
  char id[5];
  struct robot_elem *el;

  fd=fopen(ficheiro,"w");
  if (!fd) return 0;
  aux=agentes_activo;
  while(aux){
    el=(struct robot_elem *)devolve_l(aux);
    sprintf(id,"%d",el->id);
    fprintf(fd,"%s  ",id);
    fprintf(fd,"%s\n",el->nome);
    aux=aux->proximo;
  }
  fprintf(fd,"----------\n");
  aux=agentes_desactivo;
  while(aux){
    el=(struct robot_elem *)devolve_l(aux);
    sprintf(id,"%d",el->id);
    fprintf(fd,"%s  ",id);
    fprintf(fd,"%s",el->nome);
    aux=aux->proximo;
    if (aux) fprintf(fd,"\n");
  }
  fclose(fd);
  return 1;
}


int grava_end_img(char *ficheiro)
{
  FILE *fd;
  char id[5];
  struct par_cam *aux;
  int i=0;
  struct lista *l,*l1;
  struct robot_elem *r,*r1;
  int niet,niet1;

  fd=fopen(ficheiro,"w");
  if (!fd) return 0;

  for(aux=partilha.par;aux;aux=aux->prox)
  { for(l=agentes_desactivo;l;l=l->proximo)
    { r=l->info;
      if(!strcmp(r->nome,aux->nome))
      { i=r->id;
        sprintf(id,"%d  ",i);
        fprintf(fd,"%s",id);
        fprintf(fd,"%s",r->nome);
        fprintf(fd,"\n");
      }
    }
    for(l=agentes_activo;l;l=l->proximo)
    { r=l->info;
      if(!strcmp(r->nome,aux->nome))
      { i=r->id;
        sprintf(id,"%d  ",i);
        fprintf(fd,"%s",id);
        fprintf(fd,"%s",r->nome);
        fprintf(fd,"\n");
      }
    }
  }
  aux=partilha.par;

  fprintf(fd,"----------");
  for(l=agentes_activo;l;l=l->proximo)
  { r=l->info;
    niet=0;
```

```
    i++;
    for(aux=partilha.par;(aux);aux=aux->prox)
    { if(!strcmp(aux->nome,r->nome)) niet=1;
    }
    if(!niet)
    { sprintf(id,"%d  ",r->id);
      fprintf(fd,"\n%s",id);
      fprintf(fd,"%s",r->nome);
    }
  }
  for(l1=agentes_desactivo;l1;l1=l1->proximo)
  { r1=l1->info;
    niet1=0;
    i++;
    for(aux=partilha.par;(aux);aux=aux->prox)
    { if(!strcmp(aux->nome,r1->nome)) niet1=1;
    }
    if(!niet1)
    { sprintf(id,"%d  ",r1->id);
      fprintf(fd,"\n%s",id);
      fprintf(fd,"%s",r1->nome);
    }
  }
  fclose(fd);
  return 1;
}


/* funcao que le o ficheiro de configuracao dos robots */
int le_ficheiro(char *ficheiro)
{
  FILE *fd;
  static char line[BUFSIZE],str1[BUFSIZE],str2[BUFSIZE],str3[BUFSIZE];
  static int num,plataforma;
  struct robot_elem *el;
  struct controlador_str *co;

  /* plataforma 0 -> socrob
     plataforma 1 -> scout */

  if (DEBUG_FICH) printf("\n----------\nFuncao le_ficheiro\n");
  fd=fopen(ficheiro,"r");
  if (fd)
    while(!feof(fd))
     { el=(struct robot_elem *)malloc(sizeof(struct robot_elem));

      fgets(line,BUFSIZE,fd);
      if (sscanf(line,"%s %d %s %s",str1,&num,str2,str3)==4)
      { if (DEBUG_FICH) printf(" (1) -> %s ",str1);
        if (DEBUG_FICH) printf(" (2) -> %d ",num);
        if (DEBUG_FICH) printf(" (3) -> %s ",str2);
        if (DEBUG_FICH) printf(" (4) ->%s \n",str3);

        if (!strcmp(str3,"scout"))
          plataforma=1;
        else
          if(!strcmp(str3,"socrob"))
          { plataforma=0;
            co=(struct controlador_str *)malloc(sizeof(struct controlador_str));
          }
          else
            { printf("ERRO: plataforma invalida %s \n",str3);
              fclose(fd);
              return 0;
            }

        if (!((num>0)&&(num<=10)))
        { printf("ERRO: numero invalido (1-10) : %d \n",num);
          fclose(fd);
          return 0;
        }
      }
      else
        { printf("ERRO: Identificador %s \n",str2);
```

```
    fclose(fd);
    return 0;
  }

el->id=num;
el->nome=malloc(10*sizeof(char));
strcpy(el->nome,str2);

fgets(line,BUFSIZE,fd);
if (sscanf(line,"%s %d",str1,&num)==2)
{ sscanf(line,"%s%d",str1,&num);
  if (DEBUG_FICH) printf("\n (1) -> %s ",str1);
  if (DEBUG_FICH) printf(" (2) -> %d \n",num);
}
else
  { printf("ERRO: Acelaracao : %d \n",num);
    fclose(fd);
    return 0;
  }

el->accel=num;

if (DEBUG_FICH) printf("\n Plataforma %d \n",plataforma);
if (plataforma)
  el->cont=NULL;
else
  { fgets(line,BUFSIZE,fd);

    if (sscanf(line,"%s %d",str1,&num)==2)
    { if (DEBUG_FICH) printf("\n (1)  -> %s ",str1);
      if (DEBUG_FICH) printf(" (2)  -> %d \n",num);
    }
    else
      { printf("ERRO: %s : kp\n",str2);
        fclose(fd);
        return 0;
      }
    co->kp=num;

    fgets(line,BUFSIZE,fd);
    if (sscanf(line,"%s %d",str1,&num)==2)
    { if (DEBUG_FICH) printf("\n (1) -> %s ",str1);
      if (DEBUG_FICH) printf("(2) -> %d \n",num);
    }
    else
      { printf("ERRO: %s : kd\n",str2);
        fclose(fd);
        return 0;
      }
    co->kd=num;

    fgets(line,BUFSIZE,fd);
    if (sscanf(line,"%s %d",str1,&num)==2)
    { if (DEBUG_FICH) printf("\n (1)  -> %s ",str1);
      if (DEBUG_FICH) printf(" (2)  -> %d \n",num);
    }
    else
      { printf("ERRO: %s : ki\n",str2);
        fclose(fd);
        return 0;
      }
    co->ki=num;

    fgets(line,BUFSIZE,fd);
    if ((sscanf(line,"%s %d",str1,&num)==2))
    { if (DEBUG_FICH) printf("\n (1) -> %s ",str1);
      if (DEBUG_FICH) printf(" (2) -> %d \n",num);
    }
    else
      { printf("ERRO: %s : kl\n",str2);
        fclose(fd);
        return 0;
      }
    co->kl=num;
```

```
          el->cont=co;
        }/* plataforma */

      el->plataforma=plataforma;
      agentes_desactivo=insere_lista(agentes_desactivo,(void *)el);

      if ((DEBUG_FICH) && (plataforma==0))
      printf("\n kp %d kd %d ki %d kl %d \n",el->cont->kp,el->cont->kd,el->cont-
>ki,el->cont->kl);
      if (DEBUG_FICH) printf("id %d nome %s  accel %d\n",el->id,el->nome,el->accel);
    }

  fclose(fd);
  return 1;
}



void trata_driver(){

  char buf[10],ficheiro[15]="space_m.pid";
  FILE *fd;
  int pid;

  escreve_porto();

  fd=fopen(ficheiro,"r");
  fgets(buf,10,fd);
  pid=atoi(buf);
  if (DEBUG) printf("\n valor do pid do driver %d \n",pid);
     kill(pid,SIGKILL);

}


/* funcao que retira um robot da lista de robots activos */
int Retira_Activo(char *nome){

 struct robot_elem *el;
 el=(struct robot_elem *)devolve_lista(agentes_activo,nome,comp);
 if (el) agentes_activo=retira_lista(agentes_activo,nome,comp);
 else return 0;
 agentes_desactivo=insere_lista(agentes_desactivo,el);

 close(Socket_Lookup[el->id]);
 Socket_Lookup[el->id]=-1;

 return 1;
}

void act_lig(int t)
{ char *chr;
  struct sockaddr_in servidor;
  struct hostent *hp;
  struct robot_elem *el;
  struct lista *aux;
  int flag;
  sigset_t sigs;

  sigemptyset(&sigs);
  sigaddset(&sigs,SIGALRM);
  pthread_sigmask(SIG_BLOCK,&sigs,NULL);

  flag=0;

  if (DEBUG) printf("\nFuncao Actualiza_Ligacoes\n\n");
  aux=agentes_desactivo;

  while(aux!=NULL)
  { el=(struct robot_elem *)devolve_l(aux);

    if (DEBUG) printf("\n id-> %d, nome %s \n",el->id,el->nome);
    chr=(char*)malloc(sizeof(el->nome)+1);
```

```
    strcpy(chr,el->nome);

    if(strcmp(chr,"\0")){
      Socket_Lookup[el->id]=socket(AF_INET,SOCK_STREAM,0);
      if(Socket_Lookup[el->id]<0){
        printf("\n erro na criacao do socket\n");
       strcpy(chr,"KO\0");
        Socket_Lookup[el->id]=-1;
      }
      else
       if(!(hp=gethostbyname(el->nome)))
         { printf("ERRO: maquina desconhecida %s\n",el->nome);
         strcpy(chr,"KO\0");
          close(Socket_Lookup[el->id]);
          Socket_Lookup[el->id]=-1;
         }
       else{
         bzero((char*)&servidor,sizeof(servidor));
         bcopy(hp->h_addr,(char*)&servidor.sin_addr,hp->h_length);
         servidor.sin_family=AF_INET;
         servidor.sin_port=htons(PORTO);
         if(connect(Socket_Lookup[el->id],(struct
sockaddr*)&servidor,sizeof(servidor))<0)
           {
             printf("ERRO: a conectar o socket stream, maquina '%s'  \n",el->nome);
             perror("");
             strcpy(chr,"KO\0");
              close(Socket_Lookup[el->id]);
              Socket_Lookup[el->id]=-1;

           } else {
               if(DEBUG) printf("ligacao com sucesso no ""canal""
[%d]\n",Socket_Lookup[el->id]);
              flag=1;
           }
       }

     } else strcpy(chr,"KO\0");

    aux=aux->proximo;
    if(strcmp(chr,"KO\0"))
      {
       if (DEBUG) printf("inserir mais um agente: %s \n",el->nome);
         Actualiza_Cont(el);
         agentes_desactivo=retira_lista(agentes_desactivo,el->nome,comp);
         agentes_activo=insere_lista(agentes_activo,(void*)el);
      }


  }/* while */

  aux=agentes_activo;
  while(aux){
        el=(struct robot_elem *)devolve_l(aux);
        if (!Ping_Command(el)){
           if(!Retira_Activo(el->nome))
             printf("ERRO: Retira_Activo \n");
           flag=1;
        }
        aux=aux->proximo;
  }

   agentes_desactivo=ordena_lista(agentes_desactivo,comp2);
   agentes_activo=ordena_lista(agentes_activo,comp2);

}


/* actualiza as comunicacoes com os robots */
void Actualiza_Ligacoes(int t,struct memoria *partilha)
{
  struct par_cam *ptr;
  pthread_t manual_broad;
  int max;
```

```
  pthread_create(&manual_broad,NULL,(void*)act_lig,(void*)&t);

  partilha->parado=1;
  sleep(1);
  broadcast(partilha);

  pthread_join(manual_broad,NULL);
  grava_end_img("ender_img");

  for(max=0,ptr=partilha->par;ptr;ptr=ptr->prox)
  { if(ptr->DIM>max) max=ptr->DIM;
  }

  if(!partilha->par)
  { printf("falha ao tentar contactar o(s) agente: nenhum agente activo...\n\n");
    sleep(TIME_OUT);
  }

  /* alocacao de memoria para os buffer da imagem */
  if(partilha->ptr1) free(partilha->ptr1);
  if(partilha->ptr2) free(partilha->ptr2);
  if(!(partilha->ptr1=malloc(max)))
  { printf("erro na alocacao de memoria\n");
    sair();
  }
  if(!(partilha->ptr2=malloc(max)))
  { printf("erro na alocacao de memoria\n");
    sair();
  }
  partilha->parado=0;
}

/* funcoes de manipulacao de parametros */

struct Param *ParamBase = NULL;
void parmOpen( char *filename )
{
  FILE *fp;
  static char line[BUF_SIZE], parm[BUF_SIZE];
  static int arg;
  struct Param *ptr;

  fp = fopen( filename, "r" );
  if ( fp ) {
    printf( "Parsing parameter file %s\n", filename );
    while ( !feof(fp) ) {
      fgets( line, BUF_SIZE, fp );
      if ( sscanf(line, "%s%d", parm, &arg) == 2 ) {
        ptr = malloc( sizeof(struct Param) );
        ptr->next = ParamBase;
        strcpy( ptr->name, parm );
        ptr->value = arg;
        ParamBase = ptr;
      }
    }
    fclose( fp );
  }
}

void parmDump( void )
{
  struct Param *ptr;

  printf("Dumping parameters read:\n");
  for ( ptr=ParamBase ; ptr ; ptr=ptr->next )
    printf( "  %s=%d\n", ptr->name, ptr->value );
}


int parmGet( char *name, int def )
{
  struct Param *ptr;
```

```
  for ( ptr=ParamBase ; ptr ; ptr=ptr->next )
    if ( strcmp(name, ptr->name) == 0 )
      return ptr->value;
  return def;
}


/* as seguintes funcoes sao utilizadas para contornar
 o problema sucessivo no bind, em relacao `a utilizacao
 do mesmo porto de comunicacao. A "martelada" consiste
 em usar portos que vao sendo incrementados num dado
 intervalo de portos pre-definido */


/* funcao que le o porto de comunicacao a usar*/
int le_porto(void){

 FILE *fd;
 char *ficheiro;
 char buf[20];

 ficheiro=(char *)calloc(20,sizeof(buf));
 strcpy(ficheiro,"porto.dat");
 fd=fopen(ficheiro,"r");
 fgets(buf,10,fd);
 PORT=atoi(buf);
 fgets(buf,10,fd);
 PORT2=atoi(buf);
 fclose(fd);


return 0;
}

/* funcao que escreve um porto de comunicacao num ficheiro */
int escreve_porto(void){

 FILE *fd;
 char *ficheiro;
 char buf[20];


 ficheiro=(char *)calloc(20,sizeof(buf));
 strcpy(ficheiro,"porto.dat");
 fd=fopen(ficheiro,"w");
 if (PORT>20300) PORT=20050;
 sprintf(buf,"%d \n",PORT);
 fprintf(fd,buf);
 if (PORT>20300) PORT=20051;
 sprintf(buf,"%d \n",PORT2);
 fprintf(fd,buf);
 fclose(fd);

return 0;
}
```

ficheiro sem.c

```
#include "sem.h"
#include "interface.h"



void erro(char *s)
{ if (DEBUG_IMAGEM) printf(s);
  exit(1);
}

void inisem(int nmax,int semmutex,int semaloc,int *mutex,int *aloc)
{ union semun init;
  if((*mutex=semget(semmutex,1,0777|IPC_CREAT))<0)
```

```
    erro("erro semget (semmutex)\n");
  if((*aloc=semget(semaloc,1,0777|IPC_CREAT))<0)
    erro("erro semget (semaloc)\n");
  init.val=1;
  if(semctl(*mutex,0,SETVAL,init)<0)
    erro("erro semctl (semmutex)\n");
  init.val=nmax;
  if(semctl(*aloc,0,SETVAL,init)<0)
    erro("erro semctl (semaloc)\n");
}

void abresem(int nmax,int semmutex,int semaloc,int *mutex,int *aloc)
{ if((*mutex=semget(semmutex,1,0777|IPC_EXCL))<0)
    erro("erro semget (semmutex)\n");
  if((*aloc=semget(semaloc,1,0777|IPC_EXCL))<0)
    erro("erro semget (semaloc)\n");
}

void esperarmutex(key_t semid,int semnum)
{ struct sembuf s;
  s.sem_num=semnum;
  s.sem_op=-1;
  s.sem_flg=SEM_UNDO;
  if(semop(semid,&s,1)<0)
    erro("erro semop (esperarmutex)\n");
}

void assinalarmutex(key_t semid,int semnum)
{ struct sembuf s;
  s.sem_num=semnum;
  s.sem_op=1;
  s.sem_flg=SEM_UNDO;
  if(semop(semid,&s,1)<0)
    erro("erro semop (assinalarmutex)\n");
}

void esperar(key_t semid,int semnum,int uni)
{ struct sembuf s;
  s.sem_num=semnum;
  s.sem_op=-uni;
  s.sem_flg=0;
  if(semop(semid,&s,1)<0)
    erro("erro semop (esperar)\n");
}

void assinalar(key_t semid,int semnum,int uni)
{ struct sembuf s;
  s.sem_num=semnum;
  s.sem_op=uni;
  s.sem_flg=0;
  if(semop(semid,&s,1)<0)
    erro("erro semop (assinalar)\n");
}
```

ficheiro space_m.c

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Xos.h>
#include <X11/Xatom.h>
#include <X11/keysym.h>
#include "space_mouse.h"
#include "mensagem.h"
#include "xdrvlib.h"
#include "comunica.h"
#include "interface.h"


extern int magellan_current;
extern int Socket_Lookup[MAX_LIGACOES];

void magellan (void) {
```

```
 Display *display;
 Window root, window;
 /*  double MagellanSensitivity = 1.0; */

 int screennumber,width,height;
 XEvent report;
 MagellanFloatEvent MagellanEvent;
 XComposeStatus compose;
 KeySym keysym;
 int MagellanDemoEnd = FALSE;
 char MagellanBuffer[ 256 ];
 int count;
 int b_aux,z_aux;


 b_aux=0;z_aux=0;


display = XOpenDisplay( NULL );

if ( display == NULL ){
  fprintf( stderr, "Can't open display! Exit ... \n");
  exit( -1 );
};

screennumber = DefaultScreen(display);
width  = DisplayWidth(display,screennumber);
height = DisplayHeight(display,screennumber);
root   = DefaultRootWindow(display);
window = XCreateSimpleWindow( display, root, 0,0, width/5*3,height/8, 20,
                              BlackPixel(display,screennumber),
                              WhitePixel(display,screennumber) );


/************************ Create 3D Event Types ***************************/
if ( !MagellanInit( display, window ) )
 {
  fprintf( stderr, "No driver is running. Exit ... \n" );
  exit(-1);
 };

MagellanApplicationSensitivity( display, 1 );

/************************ Main Loop ***************************************/
XSelectInput( display, window, KeyPressMask | KeyReleaseMask );

while( MagellanDemoEnd == FALSE ) {


  count=XEventsQueued(display,QueuedAfterReading);


  if (count){
    XNextEvent( display, &report );

    switch( report.type ){

    case KeyRelease : /* ... */
                      break;
    case KeyPress :
     XLookupString( (XKeyEvent*)&report, MagellanBuffer,
                    sizeof(MagellanBuffer), &keysym, &compose );
     MagellanDemoEnd = keysym == XK_F1    ;
     MagellanDemoEnd = True;
     break;

    case ClientMessage :
     switch( MagellanTranslateEvent( display, &report, &MagellanEvent, 1.0, 1.0 ) )
      {
       case MagellanInputMotionEvent :

         b_aux=(int)MagellanEvent.MagellanData[ MagellanB ];

         z_aux=(int)MagellanEvent.MagellanData[ MagellanZ ];
```

```
          if( b_aux  < -80 ){
            ordem_roda(esquerda,magellan_current);
           if (DEBUG_SPACE) printf(" x->  %d \n",b_aux);
           }

           if ( b_aux > 80 ){
            ordem_roda(direita,magellan_current);
           if (DEBUG_SPACE) printf(" x->  %d \n",b_aux);
           }

           if(z_aux>80){
            ordem_roda(tras,magellan_current);
           if (DEBUG_SPACE) printf(" z->  %d \n",z_aux);
           }

           if (z_aux<-80){
            ordem_roda(frente,magellan_current);
           if (DEBUG_SPACE) printf(" z->  %d \n",z_aux);
           }

           break;

         case MagellanInputButtonPressEvent :

          if (DEBUG_SPACE) printf("Butao pressionado %d
\n",MagellanEvent.MagellanButton);



         if ((MagellanEvent.MagellanButton>=1)&&(MagellanEvent.MagellanButton<=8)&&
               Socket_Lookup[MagellanEvent.MagellanButton]!=-1)
           magellan_current=MagellanEvent.MagellanButton;


         if (MagellanEvent.MagellanButton==9) ordem_roda(para,magellan_current);

          sprintf( MagellanBuffer, "Button pressed [%c]  ",
                   MagellanEvent.MagellanButton == 9 ? '*' :
                             '0'+MagellanEvent.MagellanButton );


          break;

         case MagellanInputButtonReleaseEvent :
          break;

         default : /* another ClientMessage event */
          break;
        };
       break;
      }; /* switch */
    }
    else {}
 };  /* While */

 MagellanClose( display );
 return ;
}  /* magellan */
```

ficheiro s_img.c

```
#include "s_img.h"
#include "comunica.h"
#include "interface.h"
#include "cam.h"
#include "sem.h"
```

```
#include "misc.h"

struct sockaddr_in servi;
int sock_Dat[MAX_LIG];
int sock_act;


extern struct Param *ParamBase;
extern struct memoria partilha;

void sair()
{ char buffer[MAX_BUFFER];
  struct msg m;

  if (DEBUG_IMAGEM) printf("\nTerminando o processo de recolha de imagem\n");
  m.id=SUSP_IMG;
  lineariza(buffer,&m,0);
  sendto(sock_Dat[sock_act],buffer,MAX_BUFFER,0,
           (struct sockaddr*)&servi,sizeof(servi));
  fclose(NULL);
  exit(0);
}

void nada()
{ signal(SIGALRM,nada);
  if (DEBUG_IMAGEM)printf("ligacao falhada...\n");
}

void broadcast(struct memoria *p)
{ int n,sai,fromlen,aux,i;
  struct hostent *hp;
  struct Param *ptr;
  struct par_cam *par_ptr;
  struct sockaddr_in CliAddr;
  char buffer[MAX_BUFFER];
  struct msg m;

  sleep(TIME_OUT);
  p->par=NULL;

  for(i=0;i<MAX_LIG;i++)
    sock_Dat[i]=-1;

  fromlen=sizeof(struct sockaddr_in);
  for(ptr=ParamBase,n=0;ptr;ptr=ptr->next,n++)
  {
    if (DEBUG_IMAGEM) printf("Esperando resposta do cliente (%s)...\n",ptr->name);
      sock_Dat[n]=socket(AF_INET,SOCK_DGRAM,0);
      if(sock_Dat[n]<0)
      { perror("Erro: (broadcast,s_img.c) criacao do socket");
        sock_Dat[n]=-1;
      } else
        { par_ptr=malloc(sizeof(struct par_cam));
        }

      bzero((char*)&CliAddr,sizeof(CliAddr));
      CliAddr.sin_family=AF_INET;
      CliAddr.sin_addr.s_addr=htonl(INADDR_ANY);
      CliAddr.sin_port=htons(PORTO_IMG_CLI);
      bind(sock_Dat[n],(struct sockaddr*)&CliAddr,sizeof(CliAddr));

      bzero((char*)&servi,sizeof(servi));
      servi.sin_family=AF_INET;
      if(!(hp=gethostbyname(ptr->name)))
      { perror("maquina desconhecida");
        sock_Dat[n]=-1;
      }
      bcopy(hp->h_addr,&servi.sin_addr.s_addr,hp->h_length);
      servi.sin_port=htons(PORTO_IMG_SER);

      m.id=INIC_COM;
      lineariza(buffer,&m,0);

      if(sendto(sock_Dat[n],buffer,MAX_BUFFER,0,(struct sockaddr*)&servi,
```

```
         sizeof(servi))<0)
      { perror("erro sendto");
        sock_Dat[n]=-1;
      }

      for(sai=0;!sai;)
      { alarm(TIME_OUT);
        if(recvfrom(sock_Dat[n],buffer,MAX_BUFFER,0,(struct sockaddr*)&servi,
          &fromlen)<0)
        { alarm(0);
          perror("erro recvfrom");
          sock_Dat[n]=-1;
          sai=1;
          aux=0;
        }
        else
        { alarm(0);
          m.corpo=par_ptr;
          deslineariza(&m,buffer);
          par_ptr->prox=p->par;
          p->par=par_ptr;
          if(m.id==INFO)
          { sai=!sai;
            strcpy(par_ptr->nome,ptr->name);
            par_ptr->N_SOCK=n;
            par_ptr->DIM=par_ptr->W*par_ptr->H*par_ptr->BYTES_PIXEL;
            par_ptr->DIM_DIV_N=(int)((par_ptr->DIM)/par_ptr->DIV_N);
          }
        }
      }
    }
  for(par_ptr=p->par;par_ptr;par_ptr=par_ptr->prox)
  { if (DEBUG_IMAGEM) printf("\nAgente \"%s\" inicializado com os seguintes
parametros:\n",par_ptr->nome);
    if (DEBUG_IMAGEM)  printf("(W: %d, H: %d, DIV_N: %d, DIM: %d, DIM_DIV_N: %d)\n",
             par_ptr->W,par_ptr->H,par_ptr->DIV_N,par_ptr->DIM,par_ptr->DIM_DIV_N);
    if (DEBUG_IMAGEM) printf("ligado ao socket [%d]\n\n",par_ptr->N_SOCK);
  }
}

void rede_mem(struct memoria *p)
{
  char buffer[MAX_BUFFER];
  struct msg m;
  int b,i,sai,fromlen=sizeof(struct sockaddr_in);

  if (DEBUG_IMAGEM) printf("iniciando tarefa de recolha da imagem da rede para a
memoria\n");

  while(!p->sair)
  {
    for(;((!p->activo)||(p->parado));)
    { sleep(1);
    }

    for(b=0;(!p->pendente)&&(!p->parado);b=!b)
    { if(b)
      { esperar(p->semaloc1,0,1);
        esperarmutex(p->mutex1,0);
        for(i=0;i<p->activo->DIV_N;i++)
        { for(sai=0;!sai;)
          { alarm(TIME_OUT);
            if(recvfrom(sock_Dat[p->activo->N_SOCK],
                        p->ptr1+(i*p->activo->DIM_DIV_N),
                        p->activo->DIM_DIV_N*sizeof(char),
                        0,(struct sockaddr *)&servi,&fromlen)<0)
              { alarm(0);
                perror("recepcao do socket");
              }
            else
            { alarm(0);
              sai=!sai;
            }
          }
```

```
          m.id=SYNC_IMG;
          lineariza(buffer,&m,0);
          sendto(sock_Dat[p->activo->N_SOCK],buffer,MAX_BUFFER,0,
                  (struct sockaddr*)&servi,sizeof(servi));
        }
        assinalarmutex(p->mutex1,0);
      }
      else
      { esperar(p->semaloc2,0,1);
        esperarmutex(p->mutex2,0);
        for(i=0;i<p->activo->DIV_N;i++)
        { for(sai=0;!sai;)
          { alarm(TIME_OUT);
            if(recvfrom(sock_Dat[p->activo->N_SOCK],
                        p->ptr2+(i*p->activo->DIM_DIV_N),
                        p->activo->DIM_DIV_N*sizeof(char),
                        0,(struct sockaddr *)&servi,&fromlen)<0)
             { alarm(0);
               perror("recepcao do socket");
             }
             else
             { alarm(0);
               sai=!sai;
             }
          }
          m.id=SYNC_IMG;
          lineariza(buffer,&m,0);
          sendto(sock_Dat[p->activo->N_SOCK],buffer,MAX_BUFFER,0,
                  (struct sockaddr*)&servi,sizeof(servi));
        }
        assinalarmutex(p->mutex2,0);
      }
    }

  }
  if (DEBUG_IMAGEM) printf("terminando tarefa de recolha da imagem da rede para a
memoria\n");
  return;
}

void mem_ecra(struct memoria *p)
{ int b;

  if (DEBUG_IMAGEM) printf("iniciando tarefa de recolha de imagem da memoria para o
ecra\n");

  while(!p->sair)
  {
    for(;((!p->activo)||(p->parado));)
    { if(p->pendente)
      { p->activo=p->pendente;
        p->pendente=NULL;
        break;
      }
      sleep(1);
    }

    p->w=winOpen(p->activo->W,p->activo->H);

    if(!p->pendente)
    { if (DEBUG_IMAGEM) printf("Ligacao a' maquina \"%s\" efectuado com os seguintes
parametros:\n",p->par->nome);
      if (DEBUG_IMAGEM) printf("W: %d, H: %d, DIV_N: %d\n",p->activo->W,p->activo-
>H,p->activo->DIV_N);
    }

    for(b=0;(!p->pendente)&&(!p->parado);b=!b)
    { if(!b)
      { esperarmutex(p->mutex1,0);
        winShowC(p->w,p->ptr1);
        assinalarmutex(p->mutex1,0);
        assinalar(p->semaloc1,0,1);
      }
      else
```

```
      { esperarmutex(p->mutex2,0);
        winShowC(p->w,p->ptr2);
        assinalarmutex(p->mutex2,0);
        assinalar(p->semaloc2,0,1);
      }
    }
    winClose(p->w);
    if(p->pendente)
    { p->activo=p->pendente;
      p->pendente=NULL;
    }

  }
  if (DEBUG_IMAGEM) printf("terminando tarefa de recolha de imagem da memoria para o
ecra\n");
  return;
}


int Suspende_Imagem(){
  struct hostent *hp;
  struct msg m;
  int tam;
  char buffer[MAX_BUFFER];
  struct par_cam *ptr;

   if (DEBUG_IMAGEM) printf("\n ----------\n Suspende Imagem \n");

  partilha.parado=1;
  sleep(1);

  m.corpo=NULL;
  m.id=SUSP_IMG;
  tam=lineariza(buffer,&m,0);

  for(ptr=partilha.par;ptr;ptr=ptr->prox)
  {
      bzero((char*)&servi,sizeof(servi));
      servi.sin_family=AF_INET;
      if(!(hp=gethostbyname(ptr->nome)))
      { perror("ERRO: (suspende_imagem,c_img.c) maquina desconhecida");
      }
      bcopy(hp->h_addr,&servi.sin_addr.s_addr,hp->h_length);
      servi.sin_port=htons(PORTO_IMG_SER);

     if(sendto(sock_Dat[ptr->N_SOCK],buffer,MAX_BUFFER,0,(struct
sockaddr*)&servi,sizeof(servi))<0)
       perror("");

  }

  return 1;
}

int Activa_Imagem(){
  partilha.parado=0;
  sleep(1);

return 1;
}
```

ficheiro telem_s.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/times.h>
#include <time.h>
#include <getopt.h>
```

```
#include <readline/readline.h>
#include <signal.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <pthread.h>

#include "interface.h"
#include "comunica.h"
#include "graph.h"
#include "global_s.h"

extern struct RobotVar_Str* Vars[MAX_ROBOTS];
extern int sock_telemetria;
extern int PORT2;

int sock_trans,active;


/* States Definition */
enum { GK_SEEK=0, GK_KICK, GK_ROTLEFT, GK_ROTRIGHT,
       GK_ROTLEFT2, GK_ROTRIGHT2, GK_AUTOLOCATE,
       AT_FLWBALL, AT_SRCBALL, AT_RND, AT_PREPARE, AT_PREPARE2, AT_SHOOT,
       SRC_GOAL, MOVE_TO_GOAL, AT_SHOULD_I_GO, GK2_SEEK };

static char *labels[] = {
  "GK_SEEK", "GK_KICK", "GK_ROTLEFT", "GK_ROTRIGHT",
  "GK_ROTLEFT2", "GK_ROTRIGHT2", "GK_AUTOLOCATE",
  "AT_FLWBALL", "AT_SRCBALL", "AT_RND", "AT_PREPARE", "AT_PREP2", "AT_SHOOT",
  "SRC_GOAL", "MOVE_TO_GOAL", "AT_SHOULD_I_GO", "GK2_SEEK"
};

#define state2str(state) (labels[(state)])


int trata_ordem_show(char *buf,int sock_id)
 {
   struct msg m;
   struct Telem_Str *telem;
   struct Telem2_Str *telem2;
   char string[BUFSIZE];

   int id;


     memcpy(&id,buf,sizeof(id));
     id=ntohs(id);


     if (id==ID_SEND_TELEM_SS){

       m.corpo=(struct Telem_Str *)malloc(sizeof(struct Telem_Str));
       deslineariza(&m,buf);
       telem=m.corpo;
       telem->id=ntohs(telem->id);
       telem->var_id=ntohs(telem->var_id);
       if (DEBUG_TEL) printf("\n id %d \nvar id %d\n",telem->id,telem->var_id);
       telem->control=ntohs(telem->control);
       telem->value=ntohs(telem->value);
       if (telem->control==1){
             strcpy(string,state2str(telem->value));
            Escreve_String(telem->id,telem->var_id,string);
       } else Escreve_Valor(telem->id,telem->var_id,telem->value);
       free(telem);
     } else if (id==ID_SEND_TELEM_CS){

                 m.corpo=(struct Telem2_Str *)malloc(sizeof(struct Telem2_Str));
                 deslineariza(&m,buf);
                 telem2=m.corpo;
                 telem2->id=ntohs(telem2->id);
                 telem2->var_id=ntohs(telem2->var_id);
                 Escreve_String(telem2->id,telem2->var_id,telem2->value_str);
                 free(telem2);
           } else
```

```
                if (DEBUG_TEL) printf("\n outro Id -> %d \n",id);

  return 0;
 }


void executa(){



}


/************* begin sockets serv_show ***************/

void serv_show(){

   int                 fromlen,i,j,num;
   fd_set              rmask, Rmask;
   struct sockaddr_in  NameSocket, OtherSocket;
   char buffer[30];
  /*   char nome_r[20]; */
   int argc;char **argv;

   int t;

  Inicializa_Graph();

  for(i=0;Vars[i];i++){
    if (Vars[i]->socket_id!=-1){
      for(num=0;Vars[i]->variaveis[num];num++);
      Acrescenta_Graph(argc,argv,Vars[i]->nome_robot,num);
      for(j=0;Vars[i]->variaveis[j];j++)
        Escreve_Var(i,j,Vars[i]->variaveis[j]);
      Desenha_Grelha(i,Vars[i]->nome_robot,j);
    }
  }

   signal(SIGPIPE,SIG_IGN);
   signal(SIGINT,&sair1);
   signal(SIGKILL,&sair1);

   /*    criacao dos sockets  */
   sock_telemetria=socket(AF_INET,SOCK_STREAM,0);

   if((sock_telemetria<0))
   { perror("criacao do socket");
     exit(1);
   }


   /*    inicializacao da estrutura sockaddr */
   bzero((char*)&NameSocket,sizeof(NameSocket));
   NameSocket.sin_addr.s_addr=INADDR_ANY;
   NameSocket.sin_family=AF_INET;
   NameSocket.sin_port=ntohs(PORT2);
   if (DEBUG_TEL) printf("Ligacao ao porto %d\n",PORT2);

   /*    atribuicao do nome */
   if(bind(sock_telemetria,(struct sockaddr*)&NameSocket,sizeof(NameSocket)))
   { perror("erro no bind");
     exit(1);
  }

  /* aceita 2 ligacoes em espera */
  listen(sock_telemetria,10);

  /* inicializa da mascara */
  FD_ZERO(&rmask);
  FD_SET(sock_telemetria,&rmask);

  fromlen=sizeof(struct sockaddr_in);

  while(1){
```

```
    memcpy(&Rmask,&rmask,sizeof(rmask));

    if (select(getdtablesize(),&Rmask,NO_DESC,NO_DESC,BLOCK)<0)
    { perror("select");
      exit(1);
    }

      while((active=which(&Rmask))!=-1) {
         if(active==sock_telemetria)
         {  /* aceitacao de ligacao de um socket stream */
            sock_trans=accept(sock_telemetria,(struct
sockaddr*)&OtherSocket,&fromlen);
          FD_SET(sock_trans,&rmask);
           if (DEBUG_TEL) printf(" (Telem_a.c) Socket criado no porto
%d\n",sock_trans);
          } else
            { /*  recepcao de um socket */
             printf(" antes do read \n");
             printf("\n \n");
             if (t=read(active,buffer,30)>0)
              {
                            printf(" var %d\n",t);
               /*printf(" dentro do if \n"); */
               trata_ordem_show(buffer,active);
              } else {
               printf(" var %d\n",t);
                 if (DEBUG_TEL) printf ("Ligacao terminada no porto %d\n", active);



                  Apaga_Graph_All();

                  close (active);
                  close(sock_trans);
                  close(sock_telemetria);
                  fclose(NULL);

                  return;

                }

             }
             FD_CLR (active, &Rmask);
         }

  } /* while (1) */

} /* end serv_show */


int which(fd_set *mask) {
  int i;
  for(i=0;i<getdtablesize();i++)
     if(FD_ISSET(i, mask)) return i;
  return -1;
}

void sair1() {
  fclose(NULL);
  printf("Saindo com ^C\n");

}
```

Ficheiros existentes no Robot

Ficheiros contendo as estruturas de dados e protótipos

comunica.h

idêntico ao existente na aplicação

ficheiro global_c.h

```
#include <stdlib.h>
#include <math.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/times.h>
#include <time.h>
#include <getopt.h>
#include <readline/readline.h>
#include <signal.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <pthread.h>



#define MAX_VARIAVEIS 20
#define TAMANHO_NOME 30
#define MAX_ROBOTS 10
#define DEBUG 0
#define N_OPERACOES  6
#define BLOCK        0
#define NO_DESC      0


#define BUF_TELEM_STR 120


struct Kontrol_Str{
   short id;
   char nome_var[TAMANHO_NOME];
   short tempo;
};


struct Telem_Str{
   short id;
   short var_id;
   short control;
   short value;
};


struct Telem2_Str{
   short id;
   short var_id;
   char value_str[BUF_TELEM_STR];
};



struct Vars_Str{

  int id;
  char *variaveis[MAX_VARIAVEIS];
  int *tempo[MAX_VARIAVEIS];
  int *time_out[MAX_VARIAVEIS];

};
```

```
/* protos */


   /* Kontrol */

  void  Inicializa_Vars_Str();
  void Acrescenta_Var(int ,char *, int );
  int trata_ordem_kontrol(char *,int );


   /* Telem */

  int escreve(int );
  int Inicializa_Socket (char *);
  void processa();
  void dump_structure_c();

  /* All  */

  int which(fd_set *);
  void sair();
```

Ficheiros de código

ficheiro comunica.c

idêntico ao ficheiro existente na aplicação principal

ficheiro slave_rob.c

```
/*
 * Includes
 * ~~~~~~~~
 */


#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/times.h>
#include <time.h>
#include <getopt.h>
#include <readline/readline.h>
#include <signal.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>

#include "comunica.h"
#include "global.h"

#define DEBUG 1

#define MAX_PARM     8
```

```
#define VEL_BASE     25000
#define N_OPERACOES  6
#define BLOCK        0
#define NO_DESC      0

typedef struct vel_no{
  int vele,veld;
} vel_str;


struct ctrl_msg *ctrl_val;

vel_str* vel;

int WZ=0,VMC=0;
int IncW=5000,IncV=5000;
int pid,IsSocRob;
int sock_lig,active;

int calcula_esq_vel(int wz,int vmc){
  return wz+vmc;
}

int calcula_dir_vel(int wz,int vmc){
  return vmc-wz;
}

vel_str* actualiza_vel(vel_str* v,enum ordem  ord)
{
  switch(ord)
  { case frente:
              if(IsSocRob) VMC-=IncV; else VMC+=IncV;
         break;
    case tras:
              if(IsSocRob) VMC+=IncV; else VMC-=IncV;
         break;
    case esquerda:
              if(IsSocRob) WZ-=IncW; else  WZ+=IncW;
         break;
    case direita:
              if(IsSocRob) WZ+=IncW; else  WZ-=IncW;
         break;
    case para: VMC=0; WZ=0;
         break;
    case roda_esq:
              if(IsSocRob) WZ-=IncW; else WZ+=IncW;
              VMC=0;
         break;
    case roda_dir:
              if(IsSocRob) WZ+=IncW; else WZ-=IncW;
              VMC=0;
         break;
    default:
         break;
  }

  v->vele=calcula_dir_vel(WZ,VMC);
  v->veld=calcula_esq_vel(WZ,VMC);
  return v;

}

vel_str* inicializa_vel()
{ vel_str* mot=(vel_str*)malloc(sizeof(struct vel_no));
  mot->vele=0;
  mot->veld=0;
  return mot;
}

void actualiza_valores(struct ctrl_msg *cm) {

  cm->kp=ntohl(cm->kp);
  cm->kd=ntohl(cm->kd);
  cm->ki=ntohl(cm->ki);
```

```
  cm->kl=ntohl(cm->kl);
  cm->acc=ntohl(cm->acc);
  ctrl_val->kp=cm->kp;
  ctrl_val->kd=cm->kd;
  ctrl_val->ki=cm->ki;
  ctrl_val->kl=cm->kl;
  ctrl_val->acc=cm->acc;

  if (DEBUG) printf("acc: %d , kp: %d , kd: %d , ki: %d , kl: %d \n",(int)cm-
>acc,(int)cm->kp,(int)cm->kd,(int)cm->ki,(int)cm->kl);

  bbSet("motors.kp",cm->kp),
  bbSet("motors.ki",cm->ki),
  bbSet("motors.kd",cm->kd),
  bbSet("motors.il",cm->kl),
  bbSet("motors.nd", 5) ;

  motReset();
  motSetMode(1);

  motSetFilter();
  motSetAccel(ctrl_val->acc);
}

void processa_reset(){

  motReset();
  motSetMode(1);

  bbSet("motors.kp",ctrl_val->kp),
  bbSet("motors.ki",ctrl_val->ki),
  bbSet("motors.kd",ctrl_val->kd),
  bbSet("motors.il",ctrl_val->kl),
  bbSet("motors.nd", 5);

  motSetFilter();
  motSetAccel(ctrl_val->acc);

}

void inicializa_motor(void){
  motOpen();
  motReset();
  motSetMode(1);
 }


void trata_start(struct linha_str *l){

 int i,j;
 char *arg[MAX_PARM];

  printf("\n recebi a mensagem de start \n");
  printf("\n mensagem %s \n",l->linha);

  j=0;
   for (i=0;i<MAX_PARM;i++){
     arg[i]=(char *)calloc(5,sizeof(char));
     strcpy(arg[i]," ");
   }

   for (i=0;i<20;i++)
       if (l->linha[i]=='-') {
             strncpy(arg[j],&l->linha[i],3);
             j=j+1;
       }

  for (i=0;i<MAX_PARM;i++)
      printf(" string %s \n",arg[i]);

  sync();sync();sync();
  if ((pid=fork())==0){
      execlp("main"," ",arg[0],arg[1],arg[2],arg[3],arg[4],arg[5],arg[6],arg[7],NULL);
  }
```

```
   for (i=0;i<MAX_PARM;i++)
      free(arg[i]);

 printf("\n continuando esta merda \n");

}

 int trata_ordem(int s,char* b)
 {
   struct msg m;
   short *a;
   struct ctrl_msg *cm;
   struct incr_msg *inc;
   struct porto_msg *p;
   FILE *fd;
   char ficheiro[10],buf[10];

   int tam,id;

   memcpy(&id,b,sizeof(int));

  id=ntohs(id);

   if (DEBUG) printf("\n tipo de pedido -> %d\n",id);

   switch(id)
   { case ID_ORDEM: if(DEBUG) printf("ID_ORDEM\n");
                    m.corpo=(short *)malloc(sizeof(short));
                    deslineariza(&m,b);
                    a=m.corpo;
                    *a=ntohs(*a);
                    actualiza_vel(vel,(enum ordem )*a);
                    motSetVel(vel->vele,vel->veld);
                    free(m.corpo);
          break;
     case ID_CTR: if(DEBUG) printf("ID_CTR\n");
                  m.corpo=(struct ctrl_msg *)malloc(sizeof(struct ctrl_msg));
                  deslineariza(&m,b);
                  cm=m.corpo;
                  actualiza_valores(cm);
                  free(m.corpo);
          break;

     case ID_RESET: if(DEBUG) printf("ID_RESET\n");
                  processa_reset();
          break;

     case ID_SENSIBILITY: if(DEBUG) printf("ID_RESET\n");
                  m.corpo=(struct incr_msg *)malloc(sizeof(struct incr_msg));
                  deslineariza(&m,b);
                  inc=m.corpo;
                  IncW=(int)ntohs(inc->inc_w);
                  IncV=(int)ntohs(inc->inc_v);
                  free(m.corpo);
          break;

     case ID_PING: if(DEBUG) printf("PING\n");
                   m.id=ID_PING_OK;
                  m.tamanho_corpo=0;
                  m.corpo=NULL;
                  tam=lineariza(b,&m,0);
                   write(s,b,tam);
          break;

   case ID_PORTO: if (DEBUG) printf("\n ID_PORTO \n");
                   m.corpo=(struct porto_msg *)malloc(sizeof(struct porto_msg));
                   deslineariza(&m,b);
                   p=m.corpo;
                   p->porto1=ntohs(p->porto1);
                   p->porto2=ntohs(p->porto2);
                   strcpy(ficheiro,"porto.dat");
                   fd=fopen(ficheiro,"w");
                   sprintf(buf,"%d \n",(int)p->porto1);
```

```
                    fprintf(fd,buf);
                    sprintf(buf,"%d \n",(int)p->porto2);
                    fprintf(fd,buf);
                    fclose(fd);
                    free(m.corpo);
          break;


   case ID_START_GAME: if (DEBUG) printf ("START_GAME\n");
                    motClose();
                    m.corpo=(struct linha_str *)malloc(sizeof(struct linha_str));
                    deslineariza(&m,b);
                    trata_start(m.corpo);
                    free(m.corpo);
            break;
   case ID_STOP_GAME:
                    printf("\n ola \n");
                    kill(pid,SIGINT);
                    usleep(500000);
                    motOpen();
                    processa_reset();
             break;
  default: if(DEBUG) printf("Identificador invalido\n");
   }
   return 0;
 }

 int which(fd_set *mask)
 { int i;
   for(i=0;i<getdtablesize();i++)
     if(FD_ISSET(i, mask)) return i;
   return -1;
 }

void sair()
{
  close(sock_lig);
  close(active);
  fclose(NULL);
  sync();
  printf("Saindo com ^C\n");
  exit(0);
}


int main(int argc,char* argv[])
{
   int                 sock_trans,fromlen;
   fd_set              rmask, Rmask;
   char                buf[MAX_BUFFER];
   struct sockaddr_in  NameSocket, OtherSocket;

   ctrl_val=(struct ctrl_msg *)malloc(sizeof(struct ctrl_msg));

   bbInit();
   bbLoadParam("config.dat");
   inicializa_motor();

   IsSocRob=probeIsSocrob();
   vel=inicializa_vel();

   signal(SIGPIPE,SIG_IGN);
   signal(SIGINT,&sair);
   signal(SIGKILL,&sair);

   /* criacao dos sockets */
   sock_lig=socket(AF_INET,SOCK_STREAM,0);
   if(sock_lig<0)
   { perror("criacao do socket");
     exit(1);
   }

   /*  inicializacao da estrutura sockaddr */
   bzero((char*)&NameSocket,sizeof(NameSocket));
```

```
   NameSocket.sin_addr.s_addr=INADDR_ANY;
   NameSocket.sin_family=AF_INET;
   NameSocket.sin_port=ntohs(PORTO);
   if (DEBUG) printf("port %d\n",PORTO);

   /*  atribuicao do nome */
   if(bind(sock_lig,(struct sockaddr*)&NameSocket,sizeof(NameSocket))){
      perror("erro no bind");
      exit(1);
   }

  /* aceita 2 ligacoes em espera */
  listen(sock_lig,2);

  /* inicializa da mascara */
  FD_ZERO(&rmask);
  FD_SET(sock_lig,&rmask);

  fromlen=sizeof(struct sockaddr_in);
  while(1)
  { memcpy(&Rmask,&rmask,sizeof(rmask));

    if (DEBUG) printf("select\n");
    if (select(getdtablesize(),&Rmask,NO_DESC,NO_DESC,BLOCK)<0)
    { perror("select");
      exit(1);
    }
    while((active=which(&Rmask))!=-1)
    { if(active==sock_lig)
      { /* aceitacao de ligacao de um socket stream */
        sock_trans=accept(active,(struct sockaddr*)&OtherSocket,&fromlen);
        FD_SET(sock_trans,&rmask);
        if (DEBUG) printf("[%d] socket criado\n",sock_trans);
      } else

          if (read(active,buf,MAX_BUFFER)>0) {
             trata_ordem(active,buf);
           } else {
             if (DEBUG) printf ("[%d] ligacao terminada\n", active);
             FD_CLR(active,&rmask);
             close (active);
           }
      FD_CLR (active, &Rmask);
    }
  }

  motClose();
  return 0;
}
```

ficheiro kontrol_c.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/times.h>
#include <time.h>
#include <getopt.h>
#include <readline/readline.h>
#include <signal.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <pthread.h>

#include "comunica.h"
#include "global_c.h"
#include "global.h"
```

```
extern struct Vars_Str vars;
extern int STOP;
extern pthread_cond_t cond;
extern pthread_mutex_t mutex;
extern int sock_kontrol;
extern int sock_telem;

struct MicroAgent KontrolMA;
extern struct MicroAgent TelemMA;

extern struct MicroAgent visionMA;
int PORT2;


void  Inicializa_Vars_Str(){
  int i;

  for (i=0;i<MAX_VARIAVEIS;i++) {
   vars.variaveis[i]=NULL;
   vars.tempo[i]=NULL;
   vars.time_out[i]=NULL;
  }
}

void Acrescenta_Var(int id,char *nome_var, int time) {

 int j;

 for (j=0;vars.variaveis[j];j++);

 vars.id=id;
 vars.variaveis[j]=(char *)malloc(TAMANHO_NOME*sizeof(char));
 strcpy(vars.variaveis[j],nome_var);
 vars.tempo[j]=(int *)malloc(sizeof(int));
 *(vars.tempo[j])=time;
 vars.time_out[j]=(int *)malloc(sizeof(int));
 *(vars.time_out[j])=time;
}



int trata_ordem_kontrol(char *buf,int sock_id)
 {
   struct msg m;
   int tam;
   struct Kontrol_Str *k_var;
   char buffer[MAX_BUFFER];

     m.corpo=(struct Kontrol_Str *)malloc(sizeof(struct Kontrol_Str));
     deslineariza(&m,buf);

     if (m.id==ID_SEND_VAR){
       usleep(5000);
       k_var=m.corpo;
       k_var->id=ntohs(k_var->id);
       k_var->tempo=ntohs(k_var->tempo);
       Acrescenta_Var(k_var->id,k_var->nome_var,k_var->tempo);
       m.corpo=NULL;
       m.id=ID_SEND_VAR_ACK;
       tam=lineariza(buffer,&m,0);
       if(write(sock_id,buffer,tam)<0) printf("\n Erro: a escrever no socket \n");
      free(k_var);
     }

     if (m.id==START_TELEM){
       STOP=0;
     }

     if (m.id==STOP_TELEM){
       STOP=1;
     }

     if (m.id==INIT_TELEM){
```

```
       usleep(5000);
       microAgentSend(&TelemMA,0);
     }

     if (m.id==START_GAME){
        printf("\n start_game \n");

        microAgentSend(&visionMA,0);
     }

     if (m.id==STOP_GAME){
        sync();
        return 0;
     }
   return 1;
 }


void KontrolMAInit(struct MicroAgent *self){
 }

void KontrolMAEnd(struct MicroAgent *self) {

 }


void KontrolMALoop(struct MicroAgent *self){


   int                sock_trans,fromlen,active,PORT;
   fd_set             rmask, Rmask;
   struct sockaddr_in  NameSocket, OtherSocket;
   char buffer[MAX_BUFFER];
   char buf[10],ficheiro[15]="porto.dat";
   FILE *fd;


   fd=fopen(ficheiro,"r");
   fgets(buf,10,fd);
   PORT=atoi(buf);
   fgets(buf,10,fd);
   PORT2=atoi(buf);
   fclose(fd);

   Inicializa_Vars_Str();
   signal(SIGPIPE,SIG_IGN);


/*    criacao dos sockets  */
   sock_kontrol=socket(AF_INET,SOCK_STREAM,0);

   if((sock_kontrol<0))
   { perror("criacao do socket");
     exit(1);
   }

/*    inicializacao da estrutura sockaddr */
   bzero((char*)&NameSocket,sizeof(NameSocket));
   NameSocket.sin_addr.s_addr=INADDR_ANY;
   NameSocket.sin_family=AF_INET;
   NameSocket.sin_port=ntohs(PORT);
   if (DEBUG) printf("connect to port %d\n",PORT);

/*    atribuicao do nome */
   if(bind(sock_kontrol,(struct sockaddr*)&NameSocket,sizeof(NameSocket)))
   { perror("erro no bind");
     myexit();
     return;
  }

  /* aceita 10 ligacoes em espera */
  listen(sock_kontrol,2);
```

```
  /* inicializa da mascara */
  FD_ZERO(&rmask);
  FD_SET(sock_kontrol,&rmask);

  fromlen=sizeof(struct sockaddr_in);

  /*sock_trans=accept(sock_kontrol,(struct sockaddr*)&OtherSocket,&fromlen);  */


  while(1)
  {
       memcpy(&Rmask,&rmask,sizeof(rmask));

    if (select(getdtablesize(),&Rmask,NO_DESC,NO_DESC,BLOCK)<0)
    { perror("select");
      exit(1);
      }

       while((active=which(&Rmask))!=-1)
    { if(active==sock_kontrol)
    { /* aceitacao de ligacao de um socket stream */
      sock_trans=accept(active,(struct sockaddr*)&OtherSocket,&fromlen);
        FD_SET(sock_trans,&rmask);
        if (DEBUG) printf("Socket criado no canal %d \n",sock_trans);
           } else
             {  /*recepcao de um socket  */

            if (read(active,buffer,MAX_BUFFER)>0)
             {
              if (trata_ordem_kontrol(buffer,active)==0){
                     close(sock_telem);
                     close(sock_kontrol);
                    /* microAgentKill(&KontrolMA); */
                     myexit();
                    microAgentKill(&TelemMA);
                     sync();sync();sync();
                     return;}

             } else {
                if (DEBUG) printf ("Ligacao terminada no canal %d\n", active);
                   FD_CLR(active,&rmask);
                close (sock_trans);
              }

               }
                 FD_CLR (active, &Rmask);
    }
  }


 }



void KontrolInit( ){

  fclose(NULL);
  KontrolMA.init = KontrolMAInit;
  KontrolMA.loop = KontrolMALoop;
  KontrolMA.end  = KontrolMAEnd;
  KontrolMA.name = "Kontrol";
  microAgentInit( &KontrolMA );
}
```

ficheiro telem_c.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <sys/types.h>
```

```
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/times.h>
#include <time.h>
#include <getopt.h>
#include <readline/readline.h>
#include <signal.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <pthread.h>

#include "comunica.h"
#include "global_c.h"
#include "global.h"

struct MicroAgent TelemMA;

extern struct Vars_Str vars;
extern int sock_telem;
extern int STOP;
extern pthread_cond_t cond;
extern pthread_mutex_t mutex;
extern int PORT2;


void dump_structure_c(){

  int i;
    printf("\n ------------\n");
    printf("Dump structure \n");

  for(i=0;vars.variaveis[i];i++) {

          printf("    Nome da var -> %s ",vars.variaveis[i]);
          printf("    tempo -> %d \n",*vars.tempo[i]);
    }

}

int Inicializa_Socket (char *nome_serv){

  int fd;
  char *chr;
  struct sockaddr_in servidor;
  struct  hostent *hp;

 printf("\n\n\n\n");
 printf("\n porto %d \n",PORT2);

  fd=socket(AF_INET,SOCK_STREAM,0);
  if(fd<0){
        printf("\n Erro: na criacao do socket\n");
       strcpy(chr,"KO\0");
        fd=-1;
  } else
       if(!(hp=gethostbyname(nome_serv))) {
         printf("ERRO: maquina desconhecida %s\n",nome_serv);
         strcpy(chr,"KO\0");
          close(fd);
          fd=-1;
       } else{
         bzero((char*)&servidor,sizeof(servidor));
         bcopy(hp->h_addr,(char*)&servidor.sin_addr,hp->h_length);
         servidor.sin_family=AF_INET;
         servidor.sin_port=htons(PORT2);
         if(connect(fd,(struct sockaddr*)&servidor,sizeof(servidor))<0)
           { printf("ERRO: a conectar o socket stream ,maquina %s  \n",nome_serv);
           strcpy(chr,"KO\0");
           close(fd);
           fd=-1;

           } else {
               if(DEBUG) printf("ligacao com sucesso no ""canal"" [%d]\n",fd);
```

```
            }
        }

     if(!strcmp(chr,"KO\0")){
       printf("\n Erro: Na criacao do socket  (%s)\n",nome_serv);
       return -1;
      }

 return fd;
}



void processa() {

 int i;

 STOP=0;
 while (1) {
   if(!STOP){

     for (i=0;vars.variaveis[i];i++){

       if(*(vars.time_out[i])<=0){
         if (escreve(i)==0) return;
         *(vars.time_out[i])=*(vars.tempo[i]);
       } else
         *(vars.time_out[i])-=1;
      usleep(1000);
      }
     usleep(100000);
   }

 }

}


int escreve(int id_mensagem)
{
  struct msg m;
  int tam;
  static char buffer[MAX_BUFFER];
  int sock_fd;
  struct Telem_Str t;
  struct Telem2_Str t1;
  sock_fd=sock_telem;

  if((t.value=bbGet(vars.variaveis[id_mensagem],-1))==-2){
     strcpy(t1.value_str,bbGetString(vars.variaveis[id_mensagem],"n.d.\0"));
     t1.id=vars.id;
     t1.var_id=id_mensagem;
     t1.id=htons(t1.id);
     t1.var_id=htons(t1.var_id);
     m.id=ID_SEND_TELEM_CS;
     m.corpo=&t1;
     /* printf(" \n %s \n",t1.value_str); */
     /*     printf("\n entao tudo bem %d \n",strlen(t1.value_str)); */
     tam=lineariza(buffer,&m,sizeof(struct Telem2_Str));
  } else {

     t.id=vars.id;
     t.var_id=id_mensagem;
     if(!(strcmp(vars.variaveis[id_mensagem],"machine.state"))){
        t.control=1;
     } else t.control=0;
     t.id=htons(t.id);
     t.var_id=htons(t.var_id);
     t.control=htons(t.control);
     t.value=htons(t.value);
     m.id=ID_SEND_TELEM_SS;
     m.corpo=&t;
     tam=lineariza(buffer,&m,sizeof(struct Telem_Str));
```

```
  }

  if(write(sock_fd,buffer,tam)<0){
         printf("\n Erro:  a escrever no socket \n");
         return 0;
  }
  buffer[0]='\0';

  return 1;
}


int which(fd_set *mask)
 { int i;
   for(i=0;i<getdtablesize();i++)
     if(FD_ISSET(i, mask)) return i;
   return -1;
 }


void sair()
{ fclose(NULL);
  printf("Saindo com ^C\n");
  exit(0);
}


void TelemMAInit(struct MicroAgent *self){ }

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

void TelemMALoop(struct MicroAgent *self){

  char *nome_maquina="gedeao";


  usleep(500000);
  microAgentRecv(self,0);
  printf("\n ola 2 \n");

  sock_telem=Inicializa_Socket(nome_maquina);
  usleep(5000);
  processa();
}

void TelemMAEnd(struct MicroAgent *self){
 }

void TelemInit( void ) {
  TelemMA.init = TelemMAInit;
  TelemMA.loop = TelemMALoop;
  TelemMA.end  = TelemMAEnd;
  TelemMA.name = "Telem";
  microAgentInit( &TelemMA );
}
```

ficheiro s_img.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <sys/times.h>
#include <time.h>
#include <getopt.h>
#include <readline/readline.h>
#include <signal.h>
#include <unistd.h>
#include <sys/socket.h>
```

```
#include <netinet/in.h>
#include <pthread.h>
#include <netdb.h>
#include <bttv.h>
#include "global.h"
#include "comunica.h"


#define DEBUG 1

int                 sock_DGRAM;
struct sockaddr_in  servidor;
struct par_cam      par;
char                buffer[MAX_BUFFER];

extern char *cam_map;
extern struct video_mmap cam_conf;
extern int camera;

void time_out()
{ printf("Terminando o processo cliente\n");
  fclose(NULL);
  exit(0);
}

void visionLambda2(char *frameBuffer)
{ char buffer[MAX_BUFFER];
  char *f,sair;
  /* int ciclo; */
  int i,len=sizeof(struct sockaddr_in);
  struct msg m;

  f=frameBuffer;

  for (i=0;i<par.DIV_N;i++)
  { if(sendto(sock_DGRAM,f+(i*par.DIM_DIV_N),par.DIM_DIV_N*sizeof(char),0,
      (struct sockaddr *)&servidor,sizeof(servidor))<0)
    { perror("erro ao mandar o socket datagram");
/*      time_out(); */
    }
    if(recvfrom(sock_DGRAM,buffer,MAX_BUFFER,0,(struct sockaddr*)&servidor,&len)<0)
    { if(DEBUG) perror("erro no recvfrom");
/*     time_out(); */
    }
    deslineariza(&m,buffer);
    for(sair=0;!sair;)
    { switch(m.id)
      { case SUSP_IMG: printf("Comunicacao suspensa pelo servidor\n");
                       camClose();
                       if(recvfrom(sock_DGRAM,buffer,MAX_BUFFER,0,
                         (struct sockaddr*)&servidor,&len)<0)
                         { perror("erro recvfrom");
/*                           time_out(); */
                         }
                       deslineariza(&m,buffer);
                       sleep(1);
                       camOpen(par.W,par.H);
                       break;
/*        case SUSP1_IMG:printf("Comunicacao suspensa para inicio de jogo\n");
                       camClose();
                       ciclo=1;
                       while(ciclo){
                         if(recvfrom(sock_DGRAM,buffer,MAX_BUFFER,0,
                           (struct sockaddr*)&servidor,&len)<0)
                           { perror("erro recvfrom");
                             time_out();
                           }
                         deslineariza(&m,buffer);
                         if (m.id==INIC1_COM) ciclo=0;
                       }
                       printf("Reabrindo o device\n");
                       sleep(1);
                       camOpen(par.W,par.H);
                       break; */
```

```
        case INIC_COM: printf("Comunicacao re-inicializada pelo servidor\n");
                       m.id=INFO;
                       m.corpo=(void*)&par;
                       lineariza(buffer,&m,sizeof(struct par_cam));
                       if(sendto(sock_DGRAM,buffer,MAX_BUFFER,0,
                         (struct sockaddr *)&servidor,sizeof(servidor))<0)
                         { perror("erro sendto");
/*                           time_out(); */
                         }
                         i=par.DIV_N;
        case SYNC_IMG: sair=!sair;
                       break;
      }

    }
 }
}


void main()
{
  char                sair;
  struct msg          m;
  int                 len;
  /*  int                 status; */
  struct sockaddr_in  servi;

  signal(SIGPIPE,SIG_IGN);
  signal(SIGALRM,time_out);
  signal(SIGINT,time_out);
  signal(SIGKILL,time_out);
  signal(SIGQUIT,time_out);


  parmOpen("parmcli.dat");
  if(DEBUG) parmDump();

  par.DIV_N=parmGet("pacotes_frame",3);
  if(DEBUG) printf("DIV_N: %d\n",par.DIV_N);

  par.W=parmGet("largura",128);
  if(DEBUG) printf("W: %d\n",par.W);

  par.H=parmGet("altura",96);
  if(DEBUG)printf("H: %d\n",par.H);

  par.BYTES_PIXEL=parmGet("bytes_pixel",2);
  if(DEBUG)printf("DIM: %d\n",par.BYTES_PIXEL);

  par.DIM_DIV_N=(int)(par.H*par.W*par.BYTES_PIXEL/par.DIV_N);
  if(DEBUG)printf("DIM_DIV_N: %d\n",par.DIM_DIV_N);

  if(DEBUG) printf("PORTO: %d\n",PORTO_IMG_CLI);

  /* criacao dos sockets */
  if((sock_DGRAM=socket(AF_INET,SOCK_DGRAM,0))<0)
  { perror("criacao do socket");
    time_out();
  }

  bzero((char*)&servi,sizeof(servi));
  servi.sin_addr.s_addr=htonl(INADDR_ANY);
  servi.sin_family=AF_INET;
  servi.sin_port=htons(PORTO_IMG_SER);
  if(bind(sock_DGRAM,(struct sockaddr*)&servi,sizeof(servi)))
  { perror("erro bind\n");
    time_out();
  }

  printf("Esperando ligacao do servidor (imagem)...\n");

  for(sair=0;!sair;)
  { if(recvfrom(sock_DGRAM,buffer,MAX_BUFFER,0,(struct sockaddr*)&servidor,&len)<0)
    { perror("erro recvfrom");
```

```
      time_out();
    }
    deslineariza(&m,buffer);
    if(m.id==INIC_COM)
    { sair=!sair;
    }
  }
  printf("Comunicacao estabelecida com servidor\n");

  m.id=INFO;
  m.corpo=(void*)&par;
  linearíza(buffer,&m,sizeof(struct par_cam));

  if(sendto(sock_DGRAM,buffer,MAX_BUFFER,0,(struct sockaddr*)&servidor,len)<0)
  { perror("erro sendto");
    time_out();
  }
  for(sair=0;sair;)
  { if(recvfrom(sock_DGRAM,buffer,MAX_BUFFER,0,(struct sockaddr*)&servidor,
       &len)<0)
    { perror("erro recvfrom");
      time_out();
    }
    deslineariza(&m,buffer);
    if(m.id==SYNC_IMG) sair=!sair;
  }

/*  for(;;)
    if(fork()!=0)
    {
      wait(&status);
      printf("primario: o secundario morreu\n");
      printf("primario: lancando outro secundario\n");
    }
    else */
      camOpen(par.W,par.H);
      camGrabber(visionLambda2);
      fclose(NULL);
/*    } */
}
```

ficheiro main.c

```
static char *rcsid = "$Id: main.c,v 1.24 1999/07/30 02:15:33 socrob Exp $";
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <signal.h>

#include "global.h"
#include "global_c.h"

struct Blackboard blackboard;

extern struct MicroAgent visionMA;
extern struct MicroAgent machineMA;
extern struct MicroAgent motorsMA;
extern struct MicroAgent monitorMA;
extern struct MicroAgent proxyMA;
extern struct MicroAgent KontrolMA;
extern struct MicroAgent TelemMA;


extern int USEX, USEM, USEP;


struct Vars_Str vars;
int STOP,sock_kontrol,sock_telem;
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
```

```
int Kontrol;


void handleCtrlC(int s) {
  sync();

  printf("\nCTRL-C pressed -- quitting\n");

  close(sock_kontrol);
  close(sock_telem);
  fclose(NULL);
  microAgentKill(&proxyMA);
  microAgentKill(&motorsMA);
  microAgentKill(&machineMA);
  microAgentKill(&monitorMA);
  microAgentKill(&visionMA);


 if(Kontrol){ microAgentKill(&TelemMA);
  microAgentKill(&KontrolMA);
 }



  /*  exit(0);  */
}


void myexit() {
  sync();

  microAgentKill(&proxyMA);
  microAgentKill(&motorsMA);
  microAgentKill(&machineMA);
  microAgentKill(&monitorMA);
  microAgentKill(&visionMA);

  microAgentKill(&TelemMA);
  microAgentKill(&KontrolMA);

}


int main( int argc, char **argv ) {
  int opt_mode=-1;
  int opt_my_goal=-1;

  Kontrol=0;

  sync();

  while (1) {
    int c;

    c = getopt( argc, argv, "nphm:g:sk" );
    if ( c==-1 ) break;
    switch ( c ) {
    case 'h':
      printf("Usage: %s [-npsmghk]\n"
             "  -n = no X\n"
             "  -p = no dprintf's\n"
             "  -s = no motors\n"
              "  -k = with kontrol\n"
              "  -m = mode: 0:goal-keeper, 1:defender, 2:attacker, 3:penalty-shooter\n"
              "  -g = MY goal: 1:BLUE or 2:YELLOW (!)\n"
             "  -h = this help\n"
             "Colors used in window:\n"
             "  ball ......... +RED\n"
             "  my goal ...... +GREEN\n"
             "  other goal ... +MAGENTA\n"
             "  wall ......... +YELLOW\n"
             "  robot ........ +CYAN\n"
             "My ID is %d\n", argv[0], getRobotId());
      return 0;
```

```
      break;
    case 'n':
      printf("X disabled.\n");
      USEX = 0;
      break;
    case 'm':
      opt_mode = atoi(optarg);
      break;
    case 'g':
      opt_my_goal = atoi(optarg);
      break;
    case 's':
      printf("Motors disabled.\n");
      USEM = 0;
      break;
    case 'p':
      printf("dprintf() facility disabled.\n");
      USEP = 0;
      break;
    case 'k':
      Kontrol=1;
      break;
    }
  }

  /* Init Blackboard */
  bbInit();
  bbSet("robot.id", getRobotId());
  bbLoadParam( "config.dat" );
  {
    char fn[BUF_SIZE];
    sprintf(fn, "%d.dat", bbGet("robot.id",0));
    bbLoadParam( fn );
  }

  /* Setup according to overriding command line options */
  if ( opt_mode != -1 )
    bbSet("machine.mode", opt_mode);
  if ( opt_my_goal != -1 )
    bbSet("game.my-goal", opt_my_goal);

  /* Launch MicroAgents */
  proxyInit();

  machineInit();
  motorsInit();
  monitorInit();
  usleep(100000); /* wait 100ms before starting the wheel... */
  visionInit();


 if (Kontrol){
      TelemInit();
      KontrolInit();
 } else {
      usleep(500000);
      microAgentSend(&visionMA,0);
 }

  /* Handle ctrl-c neatly */
  signal( SIGINT, handleCtrlC );

  /* Stay right here forever! */
  /*  select( 1, NULL, NULL, NULL, NULL ); */

  pthread_join(proxyMA.thread,NULL);
  pthread_join(motorsMA.thread,NULL);
  pthread_join(machineMA.thread,NULL);
  pthread_join(monitorMA.thread,NULL);
  pthread_join(visionMA.thread,NULL);

  if (Kontrol){  pthread_join(TelemMA.thread,NULL);
  pthread_join(KontrolMA.thread,NULL); }
  sleep(1);
```

```
  motClose();
  printf("\n exit \n");
  exit(0);
}
```