DANIEL CARLOS LANDI
FÁBIO ROBERTO DE MIRANDA
RODRIGO STEPHAN DE SOUZA TELLES

Trabalho de Formatura:

Robô Móvel para o Modelo de *Subsumption*

Faculdade de Engenharia Elétrica da
Escola Politécnica da
Universidade de São Paulo

Departamento de
Engenharia de Computação e
Sistemas Digitais

2º Semestre de 1999

PCS 0588 – Laboratório de Projeto de Formatura

Trabalho de Formatura do Curso de
Engenharia Elétrica – ênfase Computação

Alunos: Fábio Roberto de Miranda
Rodrigo Stephan de Souza Telles
Daniel Carlos Landi

Faculdade de Engenharia Elétrica da
Escola Politécnica da
Universidade de São Paulo

Departamento de
Engenharia de Computação e
Sistemas Digitais

2º Semestre de 1999

Orientador: Prof. Dr. Marco Túlio Carvalho de Andrade

_____
Fábio Roberto de Miranda
Aluno

_____
Rodrigo Stephan de Souza Telles
Aluno

_____
Daniel Carlos Landi
Aluno

_____
Prof. Dr. Marco Túlio Carvalho de Andrade
Orientador

Agradecimentos

Mais do que um agradecimento pessoal, este é um reconhecimento a todos que contribuíram para a efetivação de nosso projeto.

Ao professor ***Carlos Eduardo Cugnasca*** *pelo incentivo na escolha do tema, por tirar nossas dúvidas técnicas ao longo da definição e implementação do projeto, e pela disposição em ajudar a todos os grupos.*

Ao nosso orientador, professor ***Marco Túlio Carvalho de Andrade****, pelo apoio desde o começo do ano, pela indispensável ajuda no envio dos pedidos de doações, e por nos ter finalmente ensinado a redigir relatórios.*

À professora ***Edith Ranzini*** *por ter sido uma das primeiras (e mais fervorosas) entusiastas do projeto e pelo seu importante apoio.*

Ao senhor ***Carlos Leão****, da* ***Anacom Software****, por acreditar em nossa capacidade, por ter compartilhado a nossa visão sobre a importância da integração entre empresas e universidades, e pela doação que permitiu que o projeto tomasse forma.*

Ao professor ***André Riyuti Hirakawa*** *pelas dicas sobre tópicos da robótica.*

Ao senhor ***José Dias****, da* ***Polaroid do Brasil****, pela atenção imediata e pela doação do tão procurado sonar.*

Ao técnico ***Sérgio*** *e ao engenheiro* ***Ricardo*** *pela mobilização e organização que permitiram aos alunos frequentarem o laboratório nos fins de semana.*

Às ***pizzarias do Butantã*** *por nos terem salvado no laboratório durante vários sábados à noite.*

Aos nossos ***chefes*** *e* ***orientadores*** *de estágio por nos terem liberado durante o final do curso.*

A todos os demais colegas da turma ***Comput95*** *pelo ambiente produtivo e pelo bom humor mesmo nos momentos mais estressantes do curso.*

Finalmente, às nossas ***famílias*** *e* ***namoradas*** *pela compreensão que permitiu que nos dedicássemos ao projeto. Estaremos mais organizados para as teses de mestrado... esperamos.*

Àquele *que tem inspirado e dado forças, esteja* ***Ele*** *na forma que cada um queira.*

Sumário

Lista de Figuras

Lista de Tabelas

0. Equipe

A equipe deste trabalho de formatura foi composta pelos alunos:

- Daniel Carlos Landi - NUSP 499362 – email: `daniel.landi@poli.usp.br`
- Fábio Roberto de Miranda - NUSP 947841 – email: `fmiranda@lsi.usp.br`
- Rodrigo Stephan de Souza Telles - NUSP 1811052 – email: `r-telles@usa.net`

E o orientador do grupo foi:

- Prof Dr Marco Túlio de Carvalho de Andrade – email: `mtulio@pcs.usp.br`

1. Título

O título escolhido foi: **Robô Móvel para o Modelo de *Subsumption.***

Complementando o título oficial acima, e pelos motivos que serão descritos no item *2.2 Objetivos*, também o chamamos de **Projeto Fr&d** (lê-se *fred*).

Trata-se de um nome mais popular e adequado para a divulgação do projeto, sendo composto pelas iniciais dos nomes dos integrantes do grupo: **F**ábio, **R**odrigo **&** **D**aniel.

2. Introdução

2.1 Motivação

A década de 90 foi marcada pelo incrível barateamento de diversos componentes eletrônicos, principalmente os microprocessadores, as memórias e alguns sensores. Com a decorrente popularização dessas tecnologias, ao ponto de se tornarem *hobbies* para pessoas de várias áreas, hoje podemos contar com vasta literatura técnica, uma diversidade de microcontroladores de baixo custo, *kits* para iniciantes, exemplos de circuitos para diversas finalidades e ferramentas suficientes para se montar em casa um laboratório completo a preços razoáveis.

Aproveitando-se dessa pequena revolução, a robótica se tornou um tópico bastante popular. Existem hoje produtos voltados para todas as idades, desde carro-robôs com simples lógica de controle [*SKC*] [*Novasoft*], até *kits* microcontrolados customizáveis e programáveis [*Mekatronix*] [*Lynxmotion*] [*Johuco*]. Grupos de robótica amadora são comuns nos Estados Unidos, como por exemplo o *Dallas Personal Robotic Group* [*DPRG*], e competições de todos os níveis já são tradicionais [*AUVS*] [*Wars*].

Atualmente, o *kit* mais popular e bem sucedido é o *Lego MindStorms* [*Lego*]. Seguindo a tradição em *kits* de robótica versáteis e apoiados por parcerias importantes (como *NASA* e *National Instruments*), a Lego desenvolveu o *MindStorms* durante anos em conjunto com o *AI Labs* do *MIT*. O pacote contém um bloco central encapsulando um microcontrolador, visor LCD, canal serial e portas E/S. Diversos outros blocos estão disponíveis, como motores, caixas de engrenagem, polias, eixos, detetores de proximidade, detetores de luz, microfones, alto-falantes, sensores de calor, blocos funcionais lógicos, entre outros.

Um ambiente gráfico permite que se programe o robô através de blocos funcionais de forma fácil e interativa. O programa é então transmitido ao robô e este executa as ações de forma independente. Além do excelente material, a Lego promove competições incentivando a criatividade e divulgando os melhores trabalhos. Inúmeras “crianças” dos 8 aos 80 anos têm se dedicado a essa nova filosofia de robótica.

No Brasil, laboratórios com *kits* de iniciação à eletrônica são hoje encontrados em diversas escolas de ensino médio, principalmente nas particulares onde o poder aquisitivo dos alunos permite o financiamento de tais projetos pedagógicos. Nessas escolas, estar à par de inovações tecnológicas é um fator competitivo para atrair alunos e criar uma imagem de modernidade.

Em faculdades de engenharia elétrica e mecatrônica, existe grande preocupação em incentivar o aluno com aplicações práticas das teorias lecionadas. Além de laboratórios já equipados, existe nessas faculdades pessoal técnico com capacidade para inovar e levar adiantes projetos diferentes.

Sentindo que o momento é favorável e a receptividade de escolas e faculdades para projetos nessa área é alta, e ainda inspirados pelas soluções criativas já existentes, decidimos criar um projeto que possa ser facilmente adotado por escolas e até integrado ao currículo de jovens engenheiros e técnicos em eletrônica.

2.2 Objetivos

Pretendemos projetar uma arquitetura de robô e ambiente de programação que possa ser utilizada de forma semelhante ao *Lego MindStorms*, porém mais aberta, customizável e tecnicamente complexa. Trata-se de uma tentativa de uniformizar um pouco projetos de robótica amadora, permitindo maior divulgação, portabilidade e intercâmbio de projetos entre diferentes grupos de trabalho.

Para o *hardware*, visamos criar circuitos e peças mecânicas cujos componentes podem ser encontrados comercialmente em lojas especializadas e que sejam simples o suficiente para serem entendidos por hobistas, engenheirandos, técnicos e estudantes do ensino médio. Acreditamos que isso é fundamental para a popularização do projeto.

Não esperamos que a arquitetura de *harware* que proporemos seja a única. Várias outras soluções devem ser esperadas, por isso uma necessidade de desvincularmos o *software* do *hardware*.

Dentro deste nosso novo conceito, qualquer programa funcionaria em qualquer robô, desde que um *driver* adequado seja desenvolvido e as *APIs* (*Application Programming Interface*) sejam implementadas.

Para um arquitetura *software* nos moldes que propomos, os desafios são outros. Já existem diversos compiladores de diferentes linguagens e para vários microcontroladores. Primeiro, será necessário criar *drivers* para acessar os recursos computacionais de cada arquitetura de *hardware* em cada uma dessas combinações de linguagem e microcontrolador. Segundo, e mais difícil, será garantir que exista pelo menos um compilador totalmente grátis para cada uma dessas possibilidades.

Com isso, abrimos a discussão de uma outra característica básica de nosso projeto: ele é totalmente aberto, sem direitos autorais (apenas créditos) e com mínimas restrições de uso. Entre essas restrições, podemos citar: a necessidade de se manter as mesmas condições para quaisquer produtos ou soluções criadas a partir dessa arquitetura; e a obediência ao consenso de um *grupo* quanto a quaisquer alterações nas especificações.

Esse grupo, que deverá ainda ser formado e que estará liderando a normatização do projeto, é absolutamente livre e aberto a participação de qualquer pessoa. Não haverá nenhuma forma de remuneração, promoção individual ou direitos sobre os trabalhos contribuídos. Essa filosofia é basicamente a observada pela *OpenSource Foundation* [*Open*].

Esse conceito pode parecer radical à primeira vista, mas já é bastante discutido e comprovadamente eficaz. O caso exemplo que sempre citamos é o *Linux*. Ele foi iniciado e coordenado por *Linus Torvalds*, mas a quase totalidade da programação foi feita por milhares de pessoas espalhadas pelo mundo, trabalhando por espontânea vontade e sem qualquer retribuição, a não ser a satisfação de ter seu nome incluído na lista de contribuidores do projeto.

Qualquer pessoa, que tenha ajudado ou não na elaboração do *Linux*, tem acesso ao código fonte completo de um sistema operacional que foi testado e rechecado por brilhantes programadores que tomavam cada *bug* descoberto como uma cruzada pessoal de dedicação para erradicá-lo. O resultado foi um produto bastante estável e atendendo às mais diferentes necessidades que foram surgindo.

Nenhuma empresa capitalista da atualidade conseguiria mobilizar tantos programadores motivados ou realizar pesquisas tão completas com seus consumidores quanto um grupo como este. Qual empresa conseguiria lançar um produto na rede, observar milhares de pessoas obtendo-o no mesmo instante, receber uma lista de centenas de pequenos defeitos em poucas horas e, logo após, milhares de soluções em questão de minutos?

Uma boa discussão sobre essa nova filosofia de desenvolvimento de projetos pode ser encontrada no artigo [Raymond98a]. Interessante também observar é comportamento real dos participantes desse mundo alternativo em [Raymond98b] e em que eles se baseiam [*Open*].

Mas estamos cientes de nossa limitações. Não temos ambições desmedidas e não queremos que o projeto transmita uma imagem de prepotência ou que se torne uma das muitas pseudo-revoluções técnicas que populam a *Internet* hoje em dia. Nosso objetivo principal é reunir pessoas interessadas nessa ramo da robótica amadora que, independentemente desta ou demais disciplinas, sempre foi apreciada pelos integrantes deste grupo de trabalho de formatura.

Para tornar nosso projeto mais acessível, faremos uso de dois recursos que não são tradicionalmente encontrados em trabalhos de formatura. O primeiro é o uso de nomes menos técnicos e mais parecidos com marcas de fantasias. É o caso do nome do robô (*Fr&d*), do compilador (*ESCompiler*) e da linguagem de programação (*SOUL*), todos descritos ao longo do documento. O outro recurso, também propício a encontrar resistências, foi o emprego da língua inglesa no *software*. Como visamos colocá-lo na *Internet* em breve, o uso do inglês na interface e nos comentários nos pareceu mais correto. Uma versão em português será feita quando prestarmos contas com nossos patrocinadores.

Também esperamos expandir e reformatar este projeto para que fique mais parecido com um tutorial ou livro didático. Possivelmente ainda traduzí-lo para inglês e certamente colocá-lo na *Internet*. Estaremos entrando em contato com empresas ou instituições educacionais que se predisponham a nos conceder espaço em seus *sites*.

Este trabalho de formatura é o embrião de nosso projeto que esperamos poder concluir no seu devido tempo e independentemente dos rumos profissionais de cada um. O sucesso desse projeto dependerá inicialmente de nossa dedicação e do apoio que obtivermos de empresas em áreas relacionadas e das instituições de ensino. Quiçá em algum tempo ele tome vida própria na rede e possa incentivar cada vez mais "crianças" em seus *hobbies*.

2.3 Sobre este Documento

Este documento visa descrever nossas primeiras experiências em robótica e apresentar uma proposta inicial de arquitetura que pode ser copiada e experimentada por qualquer conhecedor de eletrônica digital. Não discutiremos nenhum outro tópico relacionado ao nosso projeto mais amplo de divulgação da robótica. Cremos que o item anterior é suficiente para conhecer nossos anseios.

Todas as informações aqui mostradas são de uso livre para quaisquer fins. Não há necessidade de se obter autorização de nenhuma das partes responsáveis, desde que os devidos créditos sejam mantidos e citados. No caso de produtos comerciais baseados neste documento, deve-se disponibilizar todas as informações de forma gratuita e informar o consumidor disto. Existem muitas maneiras de se lançar um produto comercialmente viável e que esteja totalmente detalhado para que qualquer possa implementá-lo caso queira: venda de material, componentes eletrônicos e serviços (diagramação e confecção de circuitos impressos, pré-montagem, testes, suporte, cursos, etc).

Todo o material aqui apresentado, inclusive códigos fonte, está disponível na forma digital para os interessados. Caso o nosso *site* não esteja no ar, favor contatar diretamente os integrantes do grupo ou o Laboratório de Projeto de Formatura do PCS/USP.

Este documento, incluindo diagramas e tabelas, foi originalmente redigido no Microsoft Word 97. *Os títulos de cada capítulo estão em fonte Arial (poderia ser outra sem serifa neutra), tamanho 14 e em negrito. O corpo do texto utiliza Times New Roman (poderia ser outra com serifa e legível), tamanho 10 e normal. O código de programação está em Courier New (poderia ser qualquer monoespaçada), tamanho 9 (o correto seria 85% do corpo de texto) e normal.*

*Para uma melhor diagramação e inserção dos circuitos no formato EPS (*Encapsuled PostScript*), o documento foi importado no* Adobe PageMaker 6.5. *Como era de se esperar, a formatação foi alterada significantemente. Tentamos deixar o resultado o mais parecido possível. A conversão para o formato PDF foi realizada pelo Adobe* Acrobat Distiller 3.0.

As imagens do robô foram modeladas no Kinetix 3D Studio Max v3.0 *e editadas no* Adobe Photoshop 5.0. *As fotos foram escaneadas a partir do original ou tiradas com máquinas digitais. Os circuitos elétricos foram diagramados no* OrCAD Capture v6.10.

3. Especificação Funcional do Projeto

O projeto consiste de um robô móvel e programável, com processamento, alimentação, locomoção e sensoreamento próprios. Além disso, o projeto prevê métodos próprios de programação do comportamento do robô e ferramentas para tal fim. Assim, carregando-o com o programa que determina o seu comportamento, o robô se torna totalmente independente de um computador ou do usuário.

Dentre as várias alternativas de arquiteturas de robôs, escolhemos os móveis pelo maior interesse que eles geralmente despertam entre os iniciantes em robótica. Uma analogia entre robôs móveis e "pequenos animais" é natural. A liberdade de locomoção e a possibilidade de exploração de ambientes é extremamente atrativa para a criação de comportamentos.

A escolha por um robô independente do computador objetivou: uma exploração mais completa da programação de microcontroladores, uma implementação simples de sistemas embarcados e maior facilidade para a posterior criação de sistemas colaborativos utilizando diversos robôs. Todos esses tópicos são de interesse dos integrantes do grupo.

Ainda por causa de tal escolha, não implementamos nenhuma forma de reconhecimento de imagens ou visão robótica, tarefas que necessitam de grande processamento computacional. O controle remoto do robô pelo computador através de comunicação sem fio também foi desnecessário por causa da independência entre eles.

Dentro dos objetivos mais amplos de nosso projeto, conforme discutimos em *2.2 Objetivos*, a programação deve ser feita de forma independente da implementação do *hardware* do robô. Logo, uma interface uniforme será criada para atender a todos os casos. Dependendo da filosofia de programação, uma linguagem própria deverá até mesmo ser desenvolvida.

3.1 Sistema

Dividimos o sistema em três partes: o robô propriamente dito (*hardware)*, as ferramentas que serão utilizadas para a criação e compilação de comportamentos para o robô (*software do computador*) e o código resultante que será executado no microcontrolador do robô (*software do robô*).

No diagrama da Figura 1 temos os macro blocos que compõem o sistema agrupados nas três partes citadas.

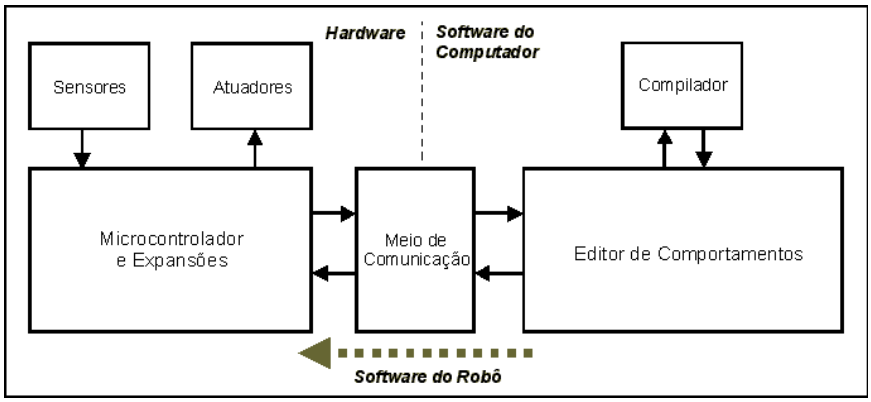


Figura 1 - Diagrama em blocos do sistema. **Créditos: Fr&d**

Uma típica programação de comportamentos teria início no *Software do Computador*, a partir do módulo *Editor de Comportamentos*, onde o usuário cria as rotinas responsáveis por executar determinadas ações. Esse comportamento em linguagem de alto nível é passado para o *Compilador* que devolve código objeto do microntrolador, compondo assim o *Software do Robô.*

O comportamento é enviado para a placa de lógica (*Microcontrolador e Expansões*) através de um *Meio de Comunicação*, de preferência serial, como RS-232, RS-485, USB, I^2C, etc. Por fim, o *Hardware* executa o comportamento, recolhendo dados do ambiente através dos *Sensores* e executando ações (tipicamente movimentos) através dos *Atuadores*.

3.2 Hardware

Em relação ao *hardware*, detalharemos aqui uma especificação que cremos ser adaptada aos nossos objetivos. Dentre as diversas alternativas possíveis de arquitetura de *hardware*, escolhemos uma que alia baixo custo, componentes relativamente populares e facilidade de implementação.

3.2.1 Microcontrolador

Uma das características principais do robô é o fato de ele ser microcontrolado. Cremos ser a melhor solução para o projeto de um robô com alguma "inteligência", mantendo o custo e a complexidade baixos. Além disso, com os recursos hoje disponíveis, pode-se praticamente criar placas de controles de robôs com apenas um CI (*robot-on-a-chip*).

Pelo custo e popularidade, existem duas alternativas: os derivados do *Intel 80C51* e os baseados no *Motorola 68HC11*. Ambos são microcontroladores de 8 bits e possuem vasta literatura e ferramentas. Optamos por um da família 8051 devido à sua maior popularidade no Brasil (versus o 68HC11 nos EUA) e disponibilidade de ferramentas adequadas no laboratório da faculdade.

A especificação pode se limitar a simplesmente esse nível de detalhe, pois é grande a compatibilidade de softwares mais simples entre os vários derivados. Como recomendação, apontamos diretamente os microcontroladores 80C51, baseados em CMOS, devido ao baixo consumo e desempenho.

O modelo de 80C51 depende da complexidade do robô e da quantidade de sensores que serão utilizados. Para robôs mais simples, os modelos mais simples de microcontroladores, com até quatro portas digitais de E/S, são suficientes. Onde houver restrição de espaço, os modelos 83Cxxx e 87Cxxx, com memória interna, economizam área e portas disponíveis.

Alguns modelos ainda possuem recursos que auxiliam o projeto do robô: saídas PWM, até três canais seriais, watchdog interno, portas digitais E/S adicionais, portas analógicas com conversor A/D embutido, timers adicionais, dois registradores DPTR, portas de captura e comparação de dados, ciclos de instrução otimizados (4 ao invés de 12), entre outros.

O uso ou não de memória externa dependerá do tamanho do programa de comportamento e da existência de memória interna no microcontrolador. Apontamos a necessidade de avaliar a real necessidade de se alocar duas portas apenas para memória externa, isto é, cabe o questionamento entre um custo maior e a vantagem de se poder utilizar mais 16 sensores.

3.2.2 Sensores

Os sensores compõem a interface entre o microcontrolador e o ambiente. É através deles que o robô se orienta e se baseia para determinar as próximas ações. Os tipos de sensores dependem do que se deseja "perceber" no ambiente. A quantidade de sensores é avaliada a partir da resolução e abrangência dos dados necessários. Para garantir medidas úteis, escolhe-se sensores com determinadas garantias de: precisão, nível de distorção, relação sinal-ruído, linearidade e sensibilidade.

Deixamos em aberto a especificação do sensores, apenas listaremos as funcionalidades necessárias seguidas de alguns exemplos [Jones&Flynn93] [Iovine98]:

- Sensor de Luminosidade: Informa ao robô a intensidade de luz em determinada direção. Com vários sensores dispostos de forma a cada um varrer um setor ao redor do robô, pode-se, por exemplo, implementar comportamentos para seguir fontes luminosas ou de se enconder em cantos escuros. Exemplos: foto-resistores, fototransistores, fotodiodos.

- Detetores de Proximidade: Emitem um sinal digital quando um obstáculo se encontra a determinada distância, permitindo que o robô desvie dele. Geralmente possuem uma sensibilidade ajustável para determinar qual a distância que irá disparar o sinal. Exemplo: detetores infravermelhos sintonizados e com *Schmitt triggers*.

- Detetores de Distância: Informam qual a distância até algum obstáculo em determinada direção. São usados principalmente para mapeamento de ambientes e para controle sobre a velocidade em robôs de alta performance. Exemplos: sonares ultrasônicos e detetores de infravermelho.

- Detetores de Som: São simplesmente microfones com alguma amplificação e opcionalmente filtrados. Permite que o robô "escute" por determinados sons ou padrões de sons, como palmas, tons de discagem, bipes e sons emitidos por outros robôs. Exemplos: microfones comuns ou de eletreto.

- Sensores de Contato: São utilizados para detetar se houve alguma colisão ou se algum membro do robô está encostando em algum material. Alguns sensores conseguem medir até mesmo a pressão do contato. Exemplos: chaves, botões, sensores de torção, fitas piezoresistivas.

- Outros sensores: São aqueles menos utilizados, mas podem ser de grande utilidade em algumas aplicações. Por exemplo: *sensores piroelétricos* que detetam fontes de calor, especialmente aquelas emitidas por mamíferos; *acelerômetros* que indicam a aceleração, podendo fornecer a distância com algumas integrações; *bússolas eletrônicas* que indicam em que direção o robô está apontando, desde que esteja em ambientes livres de interferências magnéticas; *mini-câmeras* para serem utilizadas em aplicações de tele-presença.

Todos esses sensores citados possuem custos compatíveis com o projeto que propomos. Outros sensores extremamente eficazes são ainda muito caros. Por exemplo, detetores de distância à laser com altíssima precisão (0,001%) e GPS (Global Position System) para determinação de coordenadas geográficas com precisão de algumas dezenas de metros.

3.2.3 Locomoção

Descartamos a princípio um robô que se utilize de pernas articuladas para se locomover. O custo e a complexidade não seriam compatíveis com a idéia de um projeto para iniciantes. Assim, optamos por usar um sistema baseado em rodas.

Temos várias alternativas [Iovine98] nesta classe de locomoção, como pode-se observar na *Figura 2* extraída de [Jones&Flynn93]:

- Rodas diferenciais (*differential steering*) (a): Trata-se do método mais fácil de ser implementado e programado. São duas rodas principais motorizadas, encontradas em posições diametralmente opostas, e algumas pequenas rodas para apoio.

- Configuração síncrona (*synchro drive*) (b): Utiliza três rodas equidistantes, com locomoção e tração independentes. Possui grande flexibilidade de movimentos ao custo de uma maior complexidade mecânica (por exemplo, o robô não precisa virar para mudar de direção).

- Configuração triciclo (c): Como um triciclo, três rodas, sendo uma esterçável na frente, e duas no mesmo eixo atrás.

- Configuração carro (*Ackerman steering*) (d): Similar à configuração triciclo, mas com duas rodas esterçáveis na frente e duas no mesmo eixo, atrás.

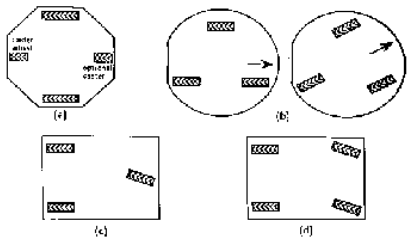


Figura 2 - Sistemas de locomoção utilizando rodas. **Créditos: [Jones&Flynn93]**

Optamos por especificar rodas diferenciais (*differential steering*) para este projeto, pois elas oferecem uma boa relação entre agilidade e facilidade de projeto e implementação. Mesmo utilizando apenas esse modo de locomoção, ainda há certa liberdade de implementação.

3.2.4 Motor

O motor a ser especificado precisava se adequar bem a restrições de peso e consumo. Consideramos duas alternativas [Jones&Flynn93], a princípio:

- Motores de passo: Sua principal vantagem é o controle simplificado, dado que podemos controlar sua posição com precisão. Quando recebe uma entrada, o motor de passo procura se estabilizar numa posição fixa. Como maior desvantagem, podemos citar o baixo torque (em relação ao seu peso). É um motor bastante adequado para mecanismos pequenos e leves (*floppy drives, winchesters*, pequenos braços mecânicos).

- Motores DC contínuos: Suas grandes vantagens são o elevado torque e alta durabilidade. Enquanto sua fonte de alimentação estiver ligada, um motor contínuo estará se movendo. Para inverter sua direção basta inverter o sentido de sua alimentação.

Optamos por especificar um motor DC contínuo. Suas características de robustez e torque *versus* peso o fazem a escolha da maioria dos projetistas de robôs móveis com rodas. Controlabilidade similar à que se obtém com motores de passo pode ser obtida com o uso de mecanismos adequados, como os *shaft encoders.*

Um *shaft encoder* é um dispositivo que consiste de um LED (em geral infravermelho), um dispositivo sensível à luz deste LED e um disco com vários setores (comumente mais que 30) pintados alternadamente de preto e branco. Seu funcionamento se baseia no fato de os setores pretos absorverem luz e os setores brancos a refletirem. A luz refletida é captada pelo dispositivo sensível à luz, que emite um pulso. Desta maneira, na saída do *shaft encoder* encontramos um trem de pulsos.

Existem também motores chamados de *DC gearhead motors* que já têm, num mesmo empacotamento, um motor DC, um *shaft encoder* e uma caixa de redução (em geral motores elétricos têm torque baixo e taxa de rotação alta, a caixa de redução abaixa a rotação para aumentar o torque).

Por fim, decidimos especificar, dentro da classe de motores contínuos, os servo-mecanismos. Cada servo-mecanismo é composto basicamente por um motor DC, um módulo de controle, um potenciômetro indicador de posição e uma caixa de redução. Tudo isto é embutido no mesmo pacote, fazendo com que o conjunto seja muito compacto.

O motor DC garante boa velocidade e força, muito mais do que um motor de passo que a princípio seria de mais fácil controle. O módulo de controle porém faz com que ele se comporte como um motor de passo, isto é, basta um sinal de referência para que o servo encontre sozinho a posição desejada. Esse sinal na verdade é na forma de PWM, que é integrado e filtrado, criando a tensão de referência.

A posição do servo aqui mencionada é o ângulo na saída da caixa de engrenagens e não no motor, pois o potenciômetro de referência está justamente ali. Aliás o próprio eixo mecânico do potenciômetro é utilizado de eixo para a engrenagem da saída da caixa de redução. A diferença entre a leitura do potenciômetro e a tensão de referência externa determina para que lado o servo deve girar.

3.2.5 Alimentação

Para que o robô seja móvel e independente, especificamos basicamente a necessidade de pilhas ou baterias como fontes de alimentação. Mesmo com essa especificação, alguns pontos importantes devem ser levantados.

Dentre os tipos de pilhas e baterias, dois são mais adequados: alcalinas e NiCad. A primeira possui alta duração (5 vezes mais que a segunda) e menor custo. Já as NiCads são recarregáveis (até 1000 vezes), permitindo até mesmo que sejam recarregadas no próprio robô. Dependendo da tolerância da tensão de entrada, pode-se até aceitar ambos os tipos.

A tensão das pilhas/baterias contudo, pode variar de acordo com o número de horas em uso [Iovine98], estado de carga, temperatura e histórico de cargas e descargas [McComb87]. Há então a necessidade de termos alguma regulação, o que pode ser feito com o uso de um simples diodo zener ou com dispositivos mais complexos, como os reguladores de tensão linear e não-linear (esse último com baixas perdas). A escolha dependerá dos requisitos do circuito a ser alimentado, da potência consumida, das perdas máximas aceitáveis, do custo e das características elétricas da saída.

3.2.6 Estrutura Mecânica

Os materiais mais comumente utilizados são: *madeira barsa*, *acrílico*, *PVC* e *alumínio.*

O uso da madeira ajuda bastante durante a prototização e resulta em um robô leve e rápido. O problema está na rigidez mecânica baixa e na progressiva deterioração.

O uso de acrílico confere um bom balanceamento de leveza, rigidez mecânica e facildade de manipulação. Por ser translúcido, facilita a checagem interior e permite que alguns sensores sejam protegidos.

O uso de chapas de alumínio resulta em um acabamento profissional, extremamente duradouro e com ótimas relações peso/rigidez. Várias montagens de precisão são viabilizadas, principalmente as conexões mecânicas de diversos dispositivos. O problema reside principalmente no custo do material e na dificuldade de se moldar as chapas. É necessária dispor de uma laboratório metalúrgico equipado com tornos, prensas, serras, etc.

Um material que não é muito conhecido, mas que se mostra bastante adequado à construção de pequenos robôs móveis, são as chapas de PVC especial. Este material é durável e resistente, além de leve e fácil de ser cortado de acordo com moldes.

Não cremos ser necessária especificar o material, ficando a escolha de acordo com a disponibilidade do material e de ferramentas.

3.3 Software do Computador

O comportamento de robô será todo determinado pelo software criado no computador. Diversos algoritmos foram contemplados [Brooks91a]: procedural, redes neurais, lógica *fuzzy*, evolutivo, *subsumption.* Destes foram escolhidos o modelo procedural e o modelo *subsumption* [Brooks86] [Brooks89] [Connell89].

O modelo *subsumption*, apesar de atualmente não ser revolucionário, é um tópico ainda novo e de interesse dos membros do grupo. Trata-se de um algoritmo robusto às falhas dos componentes, às alterações no ambiente e à problemas de lógica nos processos [Connell89]. Uma característica também importante é a *escalabilidade*, isto é, a possibilidade de incluirmos novos algoritmos sobre os existentes de maneira consistente.

No modelo de *subsumption,* o comportamento do robô é controlado por uma rede de processos que são executados em paralelo. Cada processo possui sua finalidade e tem acesso às entradas (sensores) e às saídas (motores) do robô. A comunicação entre os processos também pode ser feita utilizando tais entradas e saídas, mas observando-se certas restrições. Eles também possuem diferentes prioridades, podendo suprimir o resultado de outros processos menos prioritários quando necessário [Brooks86] [Brooks89].

Como exemplo [Brooks86], suporemos um comportamento composto dos seguintes processos, em ordem crescente de prioridade: *planejar atuação sobre o mundo, identificar objetos, monitorar mudanças, construir mapas, explorar, vagar* e *evitar objetos*. Assim, se ele estiver construindo um mapa e for necessário explorar o ambiente, o processo mais prioritário tomará conta e o robô começa a se movimentar para descobrir áreas desconhecidas. Da mesma forma, ao se deparar com um obstáculo, o robô passa a evitá-lo incodicionalmente.

Recomenda-se o estudo do *subsumption* nas referências indicadas nos parágrafos anteriores.

Uma estrutura básica para essa especificação está diagramada na *Figura 3*. O *Editor de Comportamento* é o centro de todas as ações, necessitando de comunicação com os compiladores que serão descritos e com funcionalidades para armazenamento no disco rígido do computador.

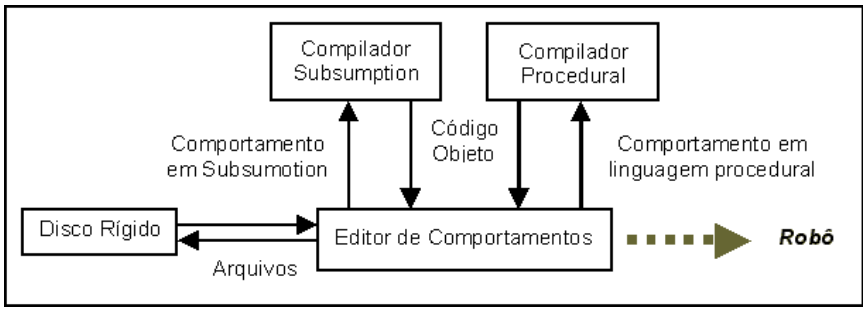


Figura 3 - Diagrama do Software do Computador. **Créditos: Fr&d**

Toda a programação deverá ser feita a partir de um computador pessoal. No caso de uma programação procedural, determinamos o uso de linguagem C devido a sua popularidade e à facilidade de se achar compiladores para os microcontroladores 8051. Com isso, elimina-se qualquer barreira quanto ao que se pode ou não fazer com o robô.

Para uma programação baseada em *subsumption*, uma linguagem própria deverá ser desenvolvida. Os conceitos básicos e funcionalidades do modelo de *subsumption* deverão ser incluídos através de estruturas e construções semânticas nessa linguagem. Um compilador para essa linguagem também faz parte desta especificação.

Para que os comportamentos sejam portáveis, haverá a necessidade de criarmos uma biblioteca de acesso aos sensors e motores. Idealmente, ao portar o código para outro robô baseado no, por exemplo, 68HC11, apenas esse *driver* será reescrito.

Alguma ferramenta para envio de comportamentos para o robô pelo canal de comunicação escolhido também é necessária.

3.4 Software do Robô

Aqui tratamos de especificar o *software* que irá ser carregado para o robô e executado pelo microcontrolador. Ele deverá ser gerado pelo *Software do Computador* e enviado pelo canal serial de comunicação escolhido e gravado na memória do robô.

Deverão existir ao menos dois módulos. O primeiro é a biblioteca de acesso aos sensores e motores. Ela deve realizar toda a interface com esses dispositivos, transformar os dados caso necessário e disponibilizar as chamadas necessárias para o segundo módulo.

Esse segundo módulo será responsável por executar toda a lógica de controle e execução do comportamento. No caso de ter sido feito por linguagem procedural, não há nenhuma estrutura básica a ser respeitada. Garantimos assim grande liberdade de “expressão” nesses programas.

No caso de um comportamento baseado em *subsumption*, surge um requisito: os diversos processos do comportamento *subsumption* devem ser executados em paralelo. Obviamente, no caso de apenas um microcontrolador, como é o *hardware* especificado, isso significa pseudo-paralelismo. O *software* deverá poder executar partes de cada processo alternadamente, compartilhando a cada momento os recursos do sistema.

Dois modos podem ser utilizados. O primeiro é o modo *cooperativo*, onde todos os processos são chamados sequencialmente e cada um deles é responsável por executador um pedaço, retornando o controle para a rotina principal. Apesar de ser mais fácil de implementar, é dependente da “boa-fé” e da estabilidade dos processos para que um deles não aloque o processamento por tempo indeterminado.

O outro é o modo *preemptivo*, onde existe um rústico sistema operacional que fornece a cada processo determinadas quantidades de tempo para alocarem os recursos do microcontrolador. Apesar de ser mais difícil de implementar, é mais robusto, confiável e requer menos complexidades dos processos que rodarão sobre ele. Uma alternativa é o uso de sistemas em tempo real já existentes para essa finalidade.

4. Diagrama em Blocos do Projeto

4.1 Hardware

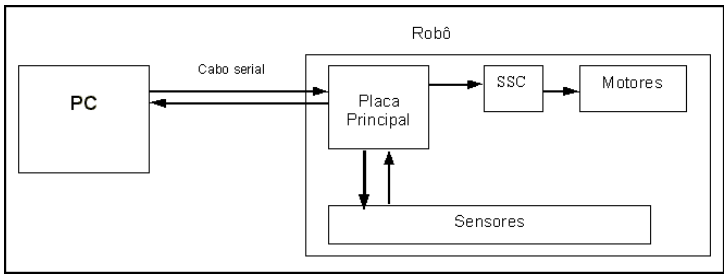


Figura 4 - Diagrama do hardware. **Créditos: Fr&d**

4.2 Software

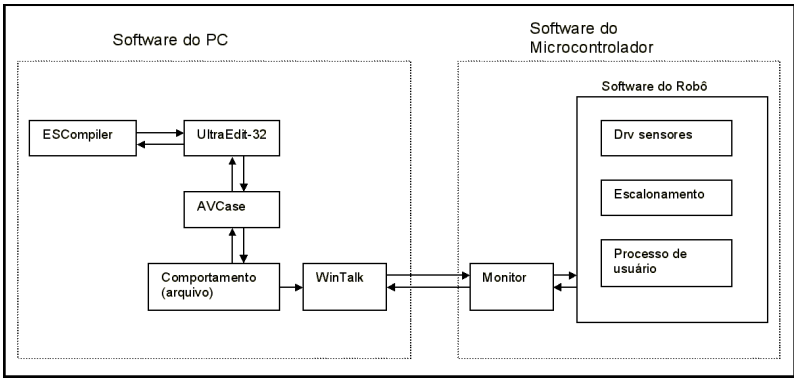


Figura 5 - Diagrama do software. **Créditos: Fr&d**

4.3 Sistema

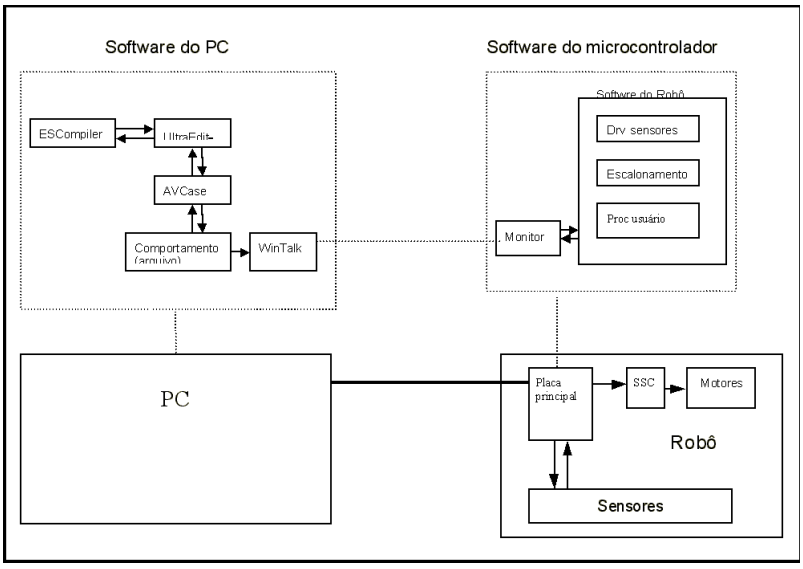


Figura 6 - Diagrama do sistema completo. **Créditos: Fr&d**

5. Descrição do Hardware

A determinação de quais equipamentos utilizaríamos foi consolidada ao recebermos o apoio da empresa *Anacom Software*. Foi a partir do *kit* que recebemos como doação que finalmente pudemos definir a estrutura de *hardware* necessária.

Tal doação tornou-se a base de nosso projeto. Foi a partir dela que construímos o resto do robô. Vários *kits* que recebemos permitiram que utilizássemos funcionalidades que antes achávamos inviáveis implementar. Uma lista do que foi doado:

- *Lynxmotion Mobile Robot Arm Kit*: base móvel com braço articulado de 5 eixos (completo, com chapa pré-cortada de PVC, 8 servos e Mini SSC II);
- *Lynxmotion Tracker Sensor Kit*: rastreador de trilha;
- *Lynxmotion Infrared Proximity Sensor*: detetor de proximidade infravermelho (4 unidades);
- *Equinox Evalu8r board*: placa de desenvolvimento com Atmel AT89S8252;
- *Equinox Activ8r programmer*: placa para programação serial do Evalu8r;
- *Keil PK51 2K*: ambiente de programação, debugação e simulação em C para 8051.

5.1 Descrição Funcional e Arquitetura

Atendendo à especificação, projetamos um robô que utiliza a base móvel da *Lynxmotion* como meio de lomocação. Nessa base, conseguimos colocar a placa principal com o 8051, canal serial e circuitos de potência, todos os sensores e ainda o braço articulado. Na base do próprio braço, existem ainda uma placa, o Mini SSC II, que controla todos os servos do robô através de comandos seriais do microcontrolador.

Decidimos optar pelo uso de memória externa, pois previmos a necessidade de reprogramação constante do comportamento. Afinal, o robô também foi projetado como um “laboratório de comportamentos”, onde o usuário pode criar diversos comportamentos e testá-los facilmente. Por isso, como poderemos ver com mais detalhes adiante, utilizamos a estrutura de memória mista onde podemos executar programas na RAM.

Para otimizar o acesso aos sensores, não mapeamos nenhum deles no espaço de memória: estão todos diretamente conectados às portas digitais e analógicas de E/S do microcontrolador.

5.1.1 Microcontrolador e Expansões

Para minimizar o número de componentes na placa de robô, foi escolhido um microcontrolador da família 80C51 que já apresente diversas expansões de funcionalidade. O modelo *Philips 80C552* é compatível com a família 80C51 e possui as seguintes características [Philips]:

- dois temporizadores/contadores padrões da família 80C51 e um outro adicional com funcionalidades de comparação de valores
- 256 bytes de RAM, expansíveis para até 64Kbytes
- dois timers padrão de 16 bits
- um conversor A/D de 10 bits e multiplexado para 8 canais
- duas saídas PWM (*pulse width modulation*) de 8 bits
- cinco portas de E/S e uma adicional apenas de entrada

- barramento de comunicação serial UART full-duplex e I^2C
- temporizador *watchdog* interno
- velocidade de 16MHz
- modos de redução de potência

Será necessário ligar uma memória EPROM ao 80C51, onde vai ficar o software mais básico, encarregado de fazer o download do programa (comportamento) a ser executado pelo robô. Para que o robô consiga executar o código que acabou de ser carregado, tínhamos duas opções: usar memória *flash* na área de programa do *80C51*, ou então usar memória de programa mapeada em RAM. Pela maior facilidade técnica envolvida, e pela necessidade constante de reprogramação dos comportamentos, decidimos usar a segunda alternativa.

A memória RAM será expandida com um 62C512 de 64Kbytes. Como o seu conteúdo terá que ser executado, utilizaremos a configuração onde o _PSEN e o _RD são conectados numa porta NAND. Assim, todo o espaço de programa e dados serão mesclados num só endereçamento. Tal expansão consumirá duas portas E/S das seis disponíveis:

- Porta 0: endereço (low) e dados da memória externa
- Porta 1: sensores digitais
- Porta 2: endereço (high) da memória externa
- Porta 3: canal serial, interrupções, contadores
- Porta 4: sensores digitais
- Porta 5: sensores analógicos

As entradas analógicas para os conversores A/D consomem a porta 5 de entrada adicional do microcontrolador e serão responsáveis por monitorar os seguintes sensores:

- Pino 5.0: foto-resistor 1
- Pino 5.1: foto-resistor 2
- Pino 5.2: foto-resistor 3
- Pino 5.3: foto-resistor 4
- Pino 5.4: foto-resistor 5
- Pino 5.5: microfone
- Pino 5.6: livre
- Pino 5.7: nível da bateria

As duas portas E/S restantes são atribuídas aos seguintes dispositivos:

- Pino P1.0: controle do detetor de proximidade infravermelho 1
- Pino P1.1: entrada 1 (QUAD_B) do detetor de proximidade infravermelho 1
- Pino P1.2: entrada 2 (QUAD_A) do detetor de proximidade infravermelho 1

- Pino P1.3: controle do detetor de proximidade infravermelho 2
- Pino P1.4: entrada 1 (QUAD_B) do detetor de proximidade infravermelho 2
- Pino P1.5: entrada 2 (QUAD_A) do detetor de proximidade infravermelho 2
- Pino P1.6: controle do detetor de proximidade infravermelho 3
- Pino P1.7: entrada 1 (QUAD_B) do detetor de proximidade infravermelho 3
- Pino P4.0: entrada 2 (QUAD_A) do detetor de proximidade infravermelho 3
- Pino P4.1: sinal 2 do *kit* seguidor de trilha
- Pino P4.2: sinal 1 do *kit* seguidor de trilha
- Pino P4.3: sinal 0 do *kit* seguidor de trilha
- Pino P4.4: chave de contato 1
- Pino P4.5: chave de contato 2
- Pino P4.6: chave de contato 3
- Pino P4.7: sonar
- Porta PWM: piezobuzzer

5.1.2 Sensores

5.1.2.1 Sensores de Colisão

Os sensores de colisão são utilizados para que, no caso de o robô se chocar com algum objeto, ele possa tomar a atitude adequada para desviar do mesmo. Normalmente estes sensores serão usados apenas em último caso, pois o robô conta com sensores de proximidade que permitem detetar um objeto antes de colidir com o mesmo.

Porém, caso o sistema de sensores infravermelho falhe ou a reação do robô responsável por desviar do obstáculo não tenha sido corretamente planejada e haja a colisão, os sensores de colisão permitem que o robô pare imediatamente o movimento que está executando para entrar no procedimento de desvio

A implementação destes sensores foi feita através de interruptores de pressão, como na *Figura 7*. O interruptor, além do botão de pressão, possui uma alavanca metálica (cerca de 6,5cm) que pressiona o botão. Isto faz com que o interruptor se torne um pouco mais sensível, permitindo que mesmo antes de haver a colisão efetivamente, o botão já tenha sido pressionado. Cinco sensores foram utilizados: dois à frente, um de cada lado e um atrás. Pára-choques cobrindo todo o perímetro do robô garantem que ao menos uma chave seja acionada em caso de qualquer contato.

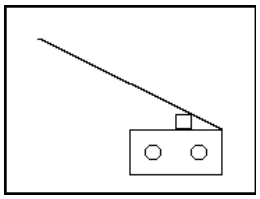


Figura 7 - Chaves de contato para deteção de colisão. **Créditos: Fr&d**

5.1.2.2 Sensores de Luminosidade

Os sensores de luminosidade foram implementados através de LDRs (resistores sensíveis à luz). A escolha do LDR para sensor luminoso se deve principalmente a sua extrema facilidade de utilização. O nível de luminosidade é convertido em tensão através de um simples divisor de tensão entre o LDR e um resistor fixo. Este nível de tensão é lido diretamente numa porta de conversão A/D do processador.

Foram utilizados 5 LDRs. A idéia é que eles fiquem distribuídos no corpo do robô de modo que a detecção das fontes de luz não dependa da orientação da base do robô.

5.1.2.3 Sensores de Proximidade

Os sensores de distância são placas *kits* da Lynxmotion baseados em raios infravermelho: o *IRPD – Infrared Proximity Detector Kit*. Os raios são emitidos por IR LEDs e sua reflexão é detetada por um receptor infravermelho do modelo Sharp GP1U52X. Dependendo da distância do objeto, o receptor acusa ou não a reflexão. Não é portanto um medidor de distância, ele apenas indica presença.

Para evitar que o sensor capte radiação infravermelho do ambiente, tanto os LEDs como o receptor estão sincronizados para operarem a 40KHz. Os LEDs emitem infravermelho em sequência de pulsos de 40KHz enquanto o detetor possui um filtro passa-banda sintonizado nessa frequência. O nível de luminosidade que dispara a saída do sensor é configurável através de um potenciômetro que integra um *Schmitt trigger*. Através do sinal diretamente extraído da saída do receptor, teríamos um sensor de proximidade ao invés de distância. [Doty96a]

Há porém um inconveniente: a reflexão do infravermelho se dá em muito menor quantidade em objetos escuros. Assim, um objeto branco ou refletivo, por exemplo, é detetado antes que um objeto preto.

Um aspecto muito interessante deste kit é quanto a sua direcionalidade. Ele possui dois emissores independentes, um de cada lado da placa, sendo que o receptor fica no meio. Fazendo com que a emissão se dê intercaladamente, entre o LED da esquerda e o LED da direita, pode-se determinar de que lado está o obstáculo.

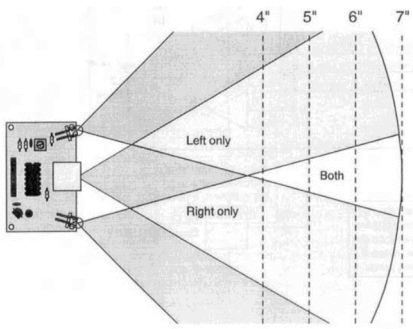


Figura 8 - Emissores e receptor do IRPD para cobrir quadrantes diferentes. **Créditos: [TSS97]**

Como se pode observar na *Figura 8*, se o objeto estiver na esquerda, por exemplo, somente será detetado durante o período em que o LED da esquerda estiver aceso. Porém, se o objeto for detetado durante os dois períodos (direita e esquerda), conclui-se que ele está no centro.

5.1.2.4 Seguidor de Trilha

O seguidor de trilha também é um *kit* da Lynxmotion: o *Tracker Kit*. Seu funcionamento é baseado em três pares de emissor/receptor de luz vermelha. Os emissores são LEDs comuns e os receptores são fototransistores

No passado, esses seguidores de trilha eram baseados em luz infravermelha. Porém, não apresentavam um desempenho constante para esse tipo de aplicação. Por isso opta-se atualmente por luz visível.

Os emissores e receptores encontram-se dois a dois juntos de forma que o receptor possa captar o reflexo no solo da luz do seu respectivo emissor. Existem duas posições possíveis para se colocar os receptores que “cercam” a fita. As posições estão indicadas na *Figura 9*.

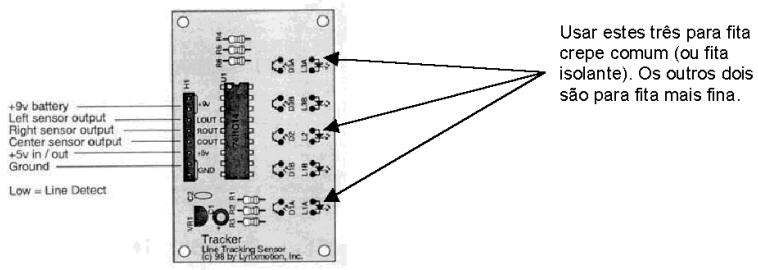


Figura 9 - Seguidor de trilha com inversores Schmitt-trigger para simplificar o circuito. **Créditos: [Lynxmotion98d]**

O que o software deve fazer é acionar alternadamente cada um dos três emissores e ao mesmo tempo monitorar cada um dos receptores. Desta forma ele sabe sempre em que posição a fita está. Se ela houver saído do centro, ela aparecerá então em um dos lados. O software deve tomar as providências para que a direção seja corrigida. Pode-se ter tanto fita preta em fundo branco, como fita branca em fundo preto.

5.1.2.5 Microfone

O microfone é um sensor muito importante para a interação com seres-humanos. Pode ser usado por exemplo para detetar uma batida de palma ou um assobio. Utilizamos um microfone de eletreto devido à sua simplicidade e custo baixo. Antes do sinal ir para o processador ele passa por um amplificador operacional e é amplificado cerca de 200 vezes. Além disso, passamos o sinal por um integrador que nos fornece um “média” da entrada do microfone nos últimos segundos. Uma palma, por exemplo, gera um alto nível de tensão por alguns segundos, tempo suficiente para ser detetado pelo microcontrolador.

5.1.3 Motor

Toda parte de movimentação do robô foi implementada com base em servo-mecanismos do tipo utilizado em modelismo: os *Hitec HS300*. São bastante populares entre os fãs de robótica devido ao seu alto desempenho. Tais servos estavam incluídos no *kit* doado pela *Anacom*.

Nesse modelo, o sinal de PWM pode ter pulsos entre 0.5ms e 2.5ms, variando a posição do servo entre o máximo à esquerda e o máximo à direita, conforme observa-se na *Figura 10*. Adiantamos aqui uma observação quanto ao valor que é passado para que o *software* defina uma posição: são números entre 0 e 255 (8 bits, 1 byte) que estão indicados entre parênteses na mesma figura.

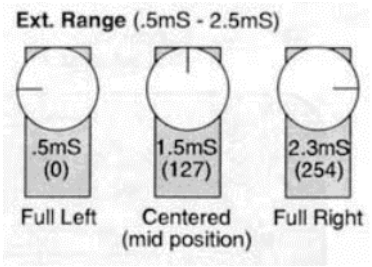


Figura 10 - Posição do servo varia entre 0° e 180° dependendo do período do PWM. **Créditos: [Lynxmotion98b]**

Os servos de posição são utilizados para a movimentação do braço do robô. Porém, para a locomoção, isto é, para os servos acoplados às rodas, existe a necessidade de se fazer uma modificação no servo. Como ele deve girar livremente, deve-se tirar o "fim de curso" do servo e substituir o potenciômetro giratório por um divisor resistivo fixo. Os detalhes desta operação estão descritos no manual [Lynxmotion98c].

Com esta modificação, o controle não é mais da posição do servo, e sim da sua velocidade. Isto é, o nível de referência determina com que velocidade o eixo estará girando (pode girar para os dois sentidos), como se pode ver na *Figura 11*.

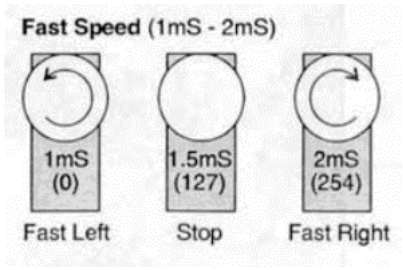


Figura 11 - Resposta ao sinal PWM do servo modificado para locomoção. **Créditos: [Lynxmotion98b]**

5.1.4 Mini SSC II (Serial Servo Controller)

Os oito servo-mecanismos utilizados no projeto são controlados por um processador PIC 16C621 dedicado que faz parte do *kit* Mini SSC II (Serial Servo Controller) [Scott]. Este circuito auxilia muito o processador principal no controle dos motores, pois ele gera os pulsos de referência necessários para os servos (*Figura 12*).

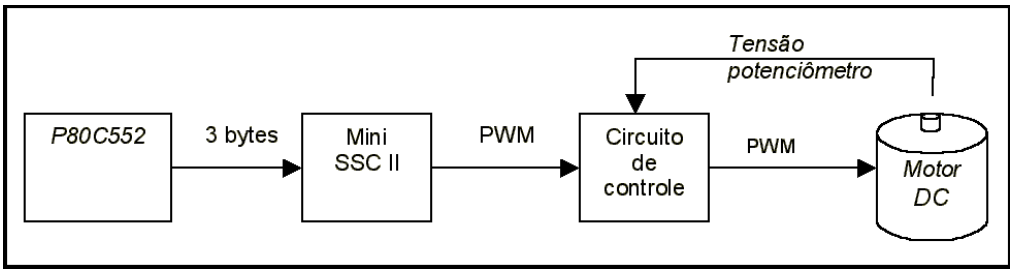


Figura 12 - Controle dos servos utilizando o Mini SSC II. **Créditos: Fr&d**

O protocolo de comunicação entre o processador principal e o PIC é muito simples. Para determinar uma nova posição para um dos servos, o processador principal envia uma seqüência de três bytes ao SSC. Como na *Tabela 1*, o primeiro byte é de sicronismo, o segundo byte indica o número do servo referenciado e, por último, o terceiro byte que indica a nova posição do servo. O SSC recebe estas informações e gera na saída correspondente (são oito ao todo) o pulso que representa a nova posição.

Tabela 1 - Pacote de comunicação com o Mini SSC II

Sincronismo (255)	Nº servo (0-7)	Posição (0-255)

O circuito pode também ter algumas configurações alteradas por jumpers (*Figura 13*):

- Abertura: 90º ou 180º. Escolhendo-se 90º os servos têm seu ângulo de trabalho diminuído, porém, a precisão é aumentada.
- Identificação: 0-7 ou 8-15. No caso de se estar trabalhando com mais de oito servos e dois SSCs.
- Taxa de transmissão: 2400 ou 9600 baud.

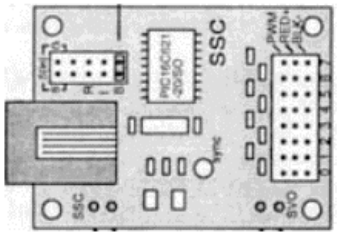


Figura 13 - A placa Mini SSC II e suas configurações por jumpers. **Créditos: [Lynxmotion98a] [Scott]**

5.1.5 Locomoção

A locomoção é baseada em duas rodas com tração independente. Há ainda uma terceira roda de apoio para equilibrar o conjunto. As rodas são de espuma sintética e semi-rígida que proporciona uma boa aderência em superfícies limpas. Já em superfícies muito empoeiradas ou úmidas, a aderência fica prejudicada. A roda traseira é bem pequena e de baixa qualidade (verifica-se muito atrito), por isso o robô fica de certa forma restrito a terrenos mais regulares.

O fato de as duas rodas de tração não possuírem eixo comum dá uma grande liberdade ao robô. Suas manobras ficam muito simples, podendo ele até mesmo girar sobre seu próprio eixo. Isto facilita os problemas de desvio de obstáculo e aumenta sua performance em locais com muitos obstáculos.

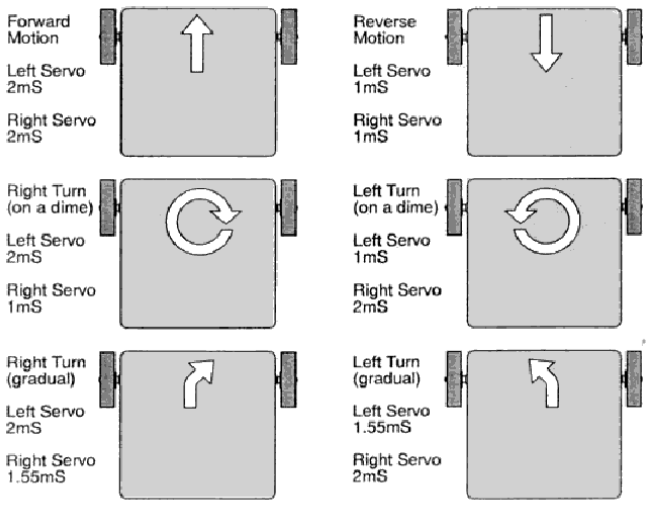


Figura 14 - Movimentos possíveis devido à configuração das rodas. **Créditos: [Lynxmotion98b]**

5.1.6 Circuitos de Potência

O robô consta de três módulos principais a serem alimentados: a placa principal com o processador P80C552 (5V), os motores (4,8 a 8,0V) e o Mini SSC II (7 a 15V), sendo que este último necessita de bateria separada pois o ruído dos motores interfere no seu funcionamento.

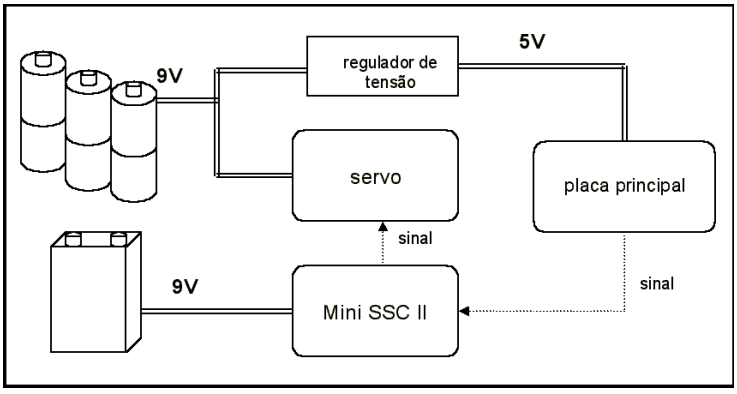


Figura 15 - Alimentação do sistema. **Créditos: Fr&d**

Foram utilizadas duas fontes de energia. Uma bateria alcalina comum de 9V para o Mini SSC II, e um conjunto de seis pilhas médias alcalinas (9V) que alimenta diretamente os motores e a placa principal através de um regulador de tensão (5V).

A escolha de pilhas alcalinas em detrimento de pilhas recarregáveis (de NiCad, por exemplo) foi feita com base na alta corrente drenada pelos motores. A pilha recarregável, apesar da vantagem econômica, possui menor carga, descarregando-se muito rapidamente. Nos momentos de maior atividade dos motores, a corrente chega a ser maior que 1A.

O regulador (linear) de tensão da placa principal é baseado no componente LM7805CV. Ele tem a propriedade de receber uma tensão maior que 7V na sua entrada e gerar 5V constantes na saída com precisão de 5%. Tudo o que excede os 5V é consumido nele e transformado em calor. Inicialmente, escolheu-se 7,5V como alimentação já que é o

mais próximo de 7V que se consegue utilizando-se pilhas de 1,5V. Porém, devido à diminuição gradual da tensão nas pilhas com o uso, adicionamos uma pilha, totalizando 9V, e garantindo tensão mesmo com os picos de corrente.

O próximo passo foi isolar o circuito do motor do circuito lógico. É muito importante fazer esta isolação porque um motor é bastante indutivo, se comportando como um curto-circuito quando faz a transição entre parado e movimentando. Isto poderia causar *spikes* que afetariam o circuito lógico, se não devidamente isolado [Jones&Flynn93].

Para filtrar os picos e evitar que este ruído se infiltre no circuito, utilizou-se capacitores de 1mF na alimentação de vários CIs. Para evitar que interferências do motor causem prejuízos à operação da parte lógica, colocamos também um diodo isolando o motor, evitando que a tensão da parte lógica caia durante a partida do motor (quando este se torna um curto).

5.1.7 Estrutura Mecânica

A maioria das peças estruturais do robô, tanto da base quando do braço, são constituídas de um tipo de PVC flexível e de cor amarela. É um material que proporciona grande facilidade de trabalho. Não é duro, por isso é fácil de cortá-lo com um estilete, por exemplo, e ao mesmo tempo tem uma grande resistência mecânica. A base do robô é feita de uma chapa de 5mm, enquanto que todas as outras peças são de uma chapa de cerca de 3mm.

Caso haja a necessidade de furação extra, uma furadeira manual é suficiente. Com relação à integração com os parafuso, este material é ideal. Ele afunda levemente quando submetido ao aperto, mantendo o parafuso sob pressão (longitudinal) e isso impede que ele se afrouxe. Porém, devido à sua baixa dureza, todos os parafusos necessitam de porca, não podendo ser afixados diretamente no material.

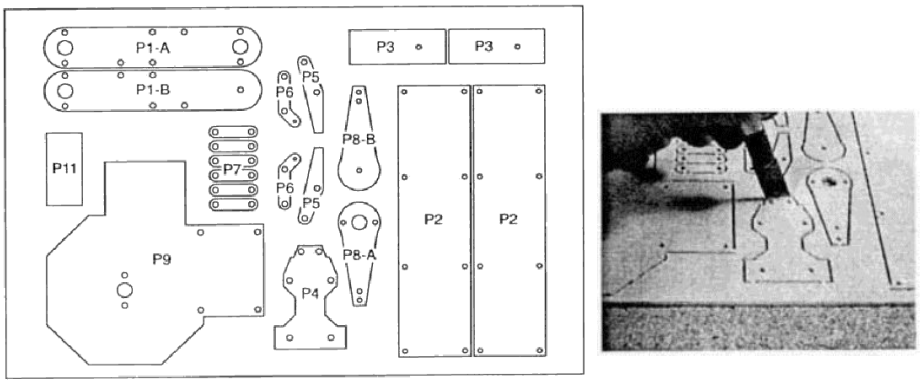


Figura 16 - Placa de PVC fornecida pelo kit com peças pré-cortadas. **Créditos: [Lynxmotion98a]**

Além deste PVC amarelo, existem ainda os espaçadores, de um plástico um pouco mais duro (cinza). São tubinhos quadrados finos. Neles é possível utilizar-se parafusos (do tipo para madeira) que se afixam diretamente na superfície interna do tubo. Eles são usados para se fixar duas chapas paralelas com um certo espaçamento entre elas.

5.1.8 Braço Articulado

O braço articulado faz parte do *kit* que grupo recebeu como doação. Apesar de não estar previsto na especificação funcional, decidimos implementá-lo e utilizá-lo por ser um item de grande apelo atrativo. Além, criou-se a chance dos membros do grupo se familiarizarem com tal tipo de dispositivo.

O braço do robô possui três segmentos e uma pinça, como podemos observar na *Figura 17*. Ao todo são cinco "articulações". São três com eixo horizontal (do tipo cotovelo), uma com eixo vertical (que permite que o braço gire) e a pinça.

O servo que determina a rotação do braço fica na parte de baixo da plataforma do robô. Seu eixo atravessa esta plataforma e é acoplado à base do braço. Como o peso do braço sempre tende para um dos lados, existem três rodinhas (cerca de 3mm) entre a plataforma do robô e a base do braço que sustentam este peso, e que permitem que o braço gire sem muito atrito.

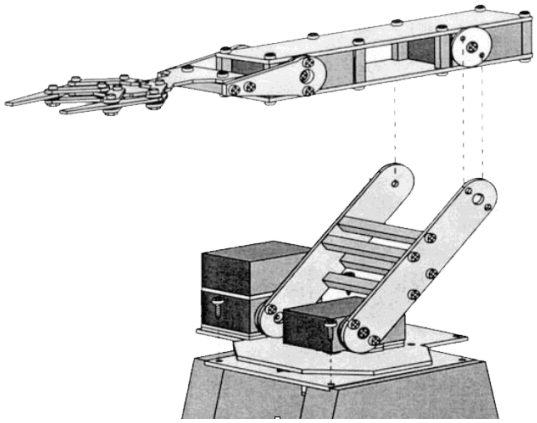


Figura 17 - O braço e suas articulações. **Créditos: [Lynxmotion98a]**

Logo em seguida, em cima da base ficam os servo-mecanismos que determinam a posição do primeiro segmento do braço. São dois servos para apenas um movimento. Isto ocorre porque este é o eixo de maior esforço. Assim os dois servos trabalham em paralelo, isto é, recebem o mesmo sinal de entrada. Por isso, é importante que eles fiquem muito bem alinhados e que suas posições iniciais sejam idênticas. Caso contrário, eles estariam trabalhando em contraposição e haveria perda de energia constante.

No segundo segmento são embutidos dois servos. Um serve para garantir seu próprio movimento em relação ao primeiro segmento e o outro é referente ao movimento do terceiro segmento (da pinça). Estes servos são presos por meio de pressão (além da fita adesiva).

No terceiro bloco, que compõe a pinça (*Figura 18*), não há outro sevo-mecanismo. O movimento de abrir e fechar da pinça é transmitido por um cabo. O servo fica colado próximo à plataforma. A transmissão da força é feita por um cabo do tipo dos de freio de bicicleta, com a única diferença de que ele age nos dois sentidos: tanto o de puxar como o de empurrar. A pinça não possui muita força tendo um pouco de dificuldade em agarrar objetos mais pesados.

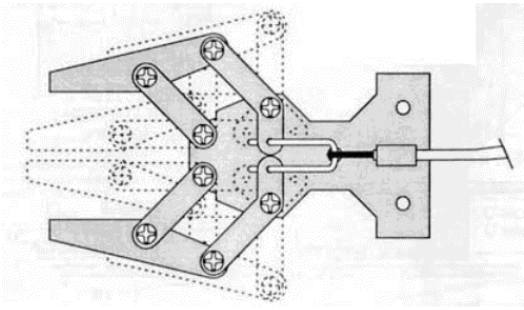


Figura 18 - A pinça do braço articulado. **Créditos: [Lynxmotion98a]**

Por um lado, o fato de não poder carregar objetos pesados é uma desvantagem mas, por outro, como existe certa flexibilidade na pinça (por isso a pequena força), não há o perigo de danos caso o fechamento da pinça seja maior que o necessário para agarrar o objeto desejado. Visto que este é um robô para demonstração (e não um utilitário), consideramos estas características favoráveis.

5.2 Esquema Elétrico

Os esquemas elétricos do projeto encontram-se em anexos. São cinco esquemas:

- Microcontrolador, Memórias e Canal Serial: *Anexo I*
- Conectores: *Anexo II*
- Sensores: *Anexo III*
- Circuito de Potência: *Anexo IV*
- Servos e Motores: *Anexo V*

5.3 Lista de Componentes

Tabela 2 - Lista de Componentes

Componente	Qte
P80C552 Philips	1
kit de braço robótico móvel Lynxmotion	1
cristal de 16 MHz	1
kit detetor de trilhas Lynxmotion	1
74HCT373	1
controlador de servomotores Lynxmotion Mini SSC II	1
RAM 62C256	1
placa de wire wrap com espaçamento pequeno (para o soquete PLCC)	1
ROM 27C512	1
seletor de 2 fios	1
barras multi-LED	3
conjuntos de conectores Molex de 5 pinos	8
chaves de contato de haste longa	6
conjunto de conectores Molex de 4 pinos	5
regulador de tensão fixa L78S05CV	1
conjuntos de conectores Molex de 3 pinos	16
transmissor / receptor CMOS – RS232	1
foto-resistores (LDR)	5
soquete PLCC de 68 pinos	1
capacitores de 33pF	2
soquetes wire wrap de 28 pinos	2
capacitore de 1nF	1
soquetes de wire wrap de 16 pinos	2
capacitores de 0.1uF	14
barras de wire wrap de 50 pinos	3
capacitor de 10uF	1
soquete de wire wrap de 20 pinos	1
capacitores eletrolíticos de 1mF	3
servomotores do tipo Hitec HS300	8
capacitores eletrolíticos de 1uF	5
kits detetor infravermelho Lynxmotion	4
capacitor eletrolítico de 10uF	1
capacitor eletrolítico de 47uF	1
conector DB9	1
74HCT00	1

5.3.1 Endereços

Visamos com este tópico facilitar ao leitor a compra de componentes eletrônicos dos tipos utilizados neste projeto. Não pretendemos favorecer nenhuma empresa e a seleção abaixo decorre apenas na nossa familiriazação com os locais tradicionais em São Paulo, a região da Santa Efigênia:

- **Compdisk** – (11) 222-8644– Rua dos Timbiras, 270
- **Gold News** – (11) 224-9500 – Rua Aurora, 82
- **Zamir** – (11) 220-3377 – Rua Sta Efigênia, 432
- **Trancham** – (11) 220-5922 – Rua Sta Efigênia, 280/519/556
- **Dual Comp** – (11) 224-8133 – Rua Aurora, 146

Entre diversos outros locais.

Para pedir um excelente catálogo:

- **Farnell do Brasil** – (11) 4066-9400 – http://www.farnell.com

5.4 Detalhes de Implementação

Procuramos descrever aqui fatos e detalhes relevantes que surgiram durante a implementação do *hardware* do robô. Não se trata de manual de montagem completo, mas dicas para ajudar aqueles que queiram repetir os mesmos passos.

5.4.1 Modificação dos servomecanismos

Para transformar o servomecanismo com ângulo de operação limitado em um com giro livre, são necessárias modificações elétricas e mecânicas. Existe um manual [Lynxmotion98c] da Lynxmotion dedicado exclusivamente a este procedimento de modificação. Seguindo-se as indicações do manual não restam muitas dúvidas. Apenas alguns probleminhas foram encontrados.

Retirar o fim de curso do potênciômetro é uma tarefa um pouco complicada porque exige força e precisão ao mesmo tempo. O fim de curso é um pedacicho da carcaça de lata entortado para dentro. Ele deve ser entortado para fora, liberando o movimento. Utilizamos uma morsa para prender o potenciômetro, e uma chave de fenda com um pequeno martelo para bater na peça de dentro para fora. Ao final deste procedimento a carcaça do potenciômetro estava entortada e teve de ser remodelada com o uso de dois alicates.

Durante a remontagem do conjunto dentro da caixinha encontramos dificuldades em encaixar tudo novamente pois alguns fios tiveram suas posições alteradas devido às mudanças elétricas. A solução foi alargar o orifício de passagem dos fios com o auxílio de um estilete.

Constatamos que esta alteração é necessária pois na primeira vez em que tentamos forçar o fechamento da caixinha com uma certa pressão, ocorreu o rompimento de soldas na plaquinha do circuito de controle.

5.4.2 Braço Articulado

O manual de montagem do braço do *kit* da *Linxmotiom* [Lynxmotion98a] é muito completo e bem ilustrado, restando poucas dúvidas sobre a montagem do braço. O primeiro ponto em que encontramos alguma insegurança foi com relação ao tipo de colagem utilizada em toda a estrutura. Utiliza-se uma fita adesiva de dupla face com um material esponjoso entre as duas faces. No início levantamos a suspeita de que estas fitas não aguentariam o esforço, mas depois de alguns testes verificamos que ela possui boa resistência.

Um ponto muito importante é o alinhamento perfeito entre os dois servo-motores que movimentam o braço em relação à base (*Figura 19*). Devido ao peso do braço, é necessário o uso de dois motores trabalhando juntos para garantir o movimento. Por isso, os dois devem ser alinhados antes de serem afixados.

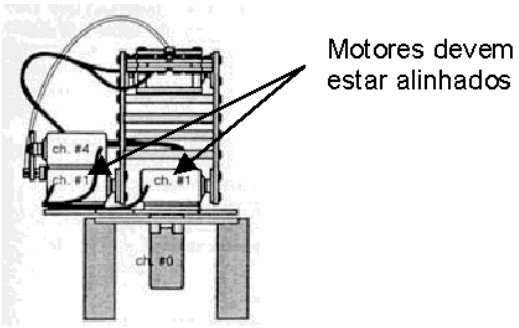


Figura 19 - Motores que devem estar em total sincronia e alinhamento. **Créditos: [Lynxmotion98a]**

Caso isto não ocorra, eles vão estar trabalhando em contraposição e gastando muito mais energia do que o necessário. O alinhamento é feito ligando-se os motores à mesma saída do SSC ainda sem os parafusos apertados, e colocar a peça de acoplamento de maneira que os dois lados do braço estejam bem paralelos. Em alguns casos, deve-se até tentar várias combinações de servos até achar dois “irmãos”.

Outra dificuldade encontrada foi com relação ao alinhamento entre o eixo de um servo-mecanismo e um parafuso. Isto acontece nos dois servos que ficam suspensos no braço. Como nesses casos existe só um motor para o movimento, deve-se complementar o eixo do outro lado com um parafuso. O alinhamento deste parafuso também deve ser muito cuidadoso. Caso eles fiquem fora do mesmo eixo, o servo ficará sobrecarregado.

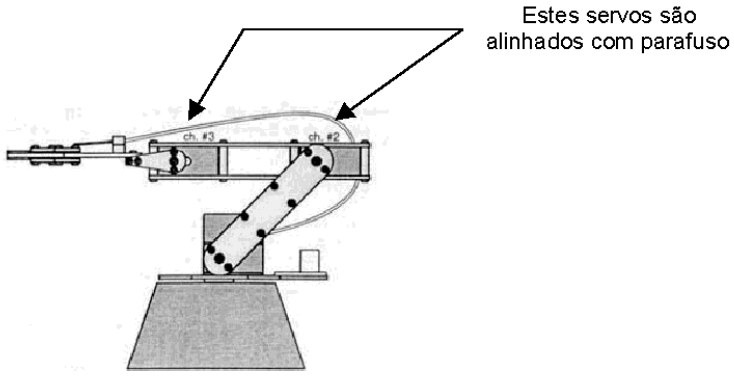


Figura 20 - Alinhamento de parafusos e servos. **Créditos: [Lynxmotion98a]**

O alinhamento foi feito manualmente, pois não encontramos nenhuma técnica melhor. O que ajuda um pouco é atarrachar um outro parafuso no eixo do servo e olhar o conjunto por muitos ângulos diferentes antes de afixar. Depois de afixado testa-se e, se for necessário, desmonta-se e monta-se novamente; assim por diante, utilizando o método da tentativa e erro.

5.4.3 Modificações na montagem original da base

Quanto à acomodação dos circuitos (placa principal e placas dos sensores) na base do robô, verificamos que haveria falta de espaço. O movimento giratório do braço impede que as placas possam ser colocadas próximas dele. Descartamos a idéia de fazer extensões na base aumentando o tamanho original, pois poderia comprometer os movimentos do robô.

A solução fui suspender o braço do robô cerca de 5cm. Para isto foram usados parafusos longos (7cm) e espaçadores feitos com o corpo de caneta *Bic*®. Cortamos quatro segmentos da caneta no comprimento exato em que o braço deveria ser suspenso. Retiramos os pequenos parafusos que uniam a base do robô à base do braço. No lugar deles colocamos os parafusos compridos atravessando os espaçadores, de modo que a base do braço ficasse distanciada da base do robô.

Com isso, foi liberado espaço na base suficiente para a colocação das placas . O circuito principal foi colocado na parte posterior do robô e os sensores na parte frontal embaixo da base do braço, agora suspensa. Uma visão parcial do robô pode ser vista na *Figura 21*.

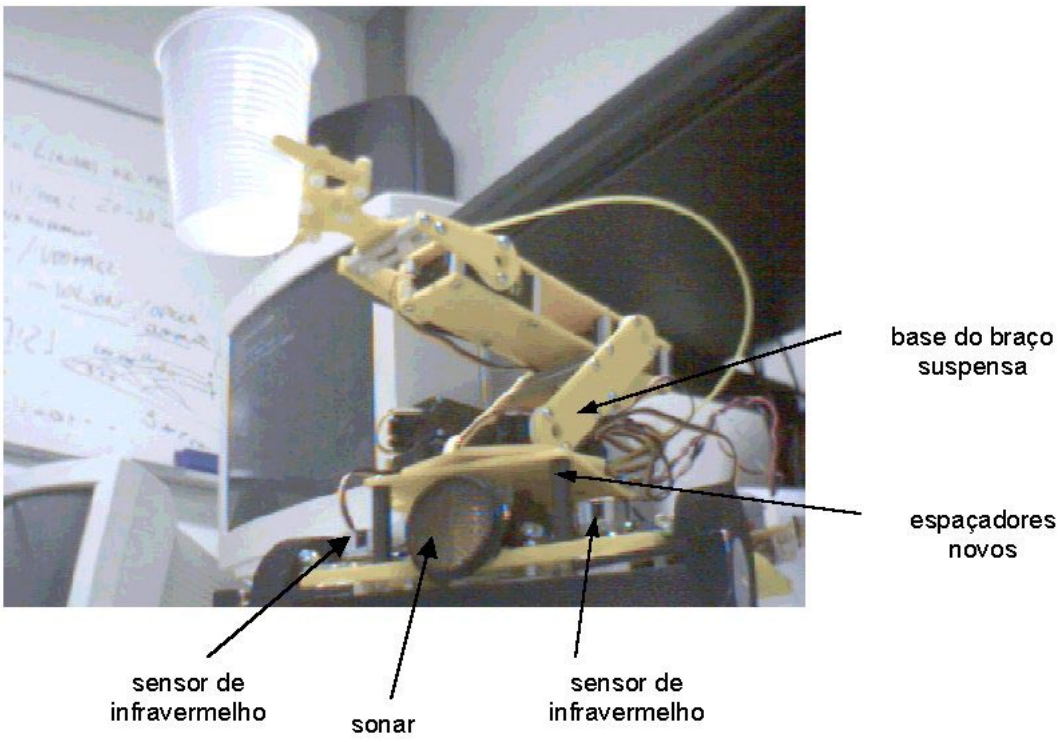


Figura 21 - Visão parcial frontal do robô, alguns sensores e a base do braço suspensa. **Créditos: Fr&d**

5.4.4 Circuito Principal

A técnica escolhida para a implementação do circuito principal do projeto foi a técnica do *Wire-Wrap*. Ela permite rápida prototipização, boa compactação do circuito e garante bons contatos nas ligações.

A preocupação com o tamanho e o fato de estarmos utilizando um processador com pinagem PLCC, nos levou a buscar uma placa de *wire-wrap* menor que a do laboratório e com o dobro da densidade de furos. Por coincidência, suas dimensões ajustavam-se exatamente à largura do robô, não implicando em necessidade de corte. A possibilidade de serem feitas soldagens na placa facilitou muito a fixação de alguns componentes, como interruptor, conectores *Molex* e chave seletora.

Os conectores *Molex* foram escolhidos por algumas vantagens em relação a outros tipos de conectores. A principal delas é a dimensão reduzida, que é importante visto a grande quantidade de conectores necessários na placa. Outras características favoráveis são: bom contato elétrico, facilidade de encaixe macho-fêmea e facilidade de montagem.

Procuramos dispor os componentes na placa de forma que os fios ficassem o mais curto possível , evitando criarmos antenas para ruídos e o mínimo de cruzamentos. Evitamos também os *closed-loops* nos barramentos de alimentação. A placa pode ser vista na *Figura 22*.

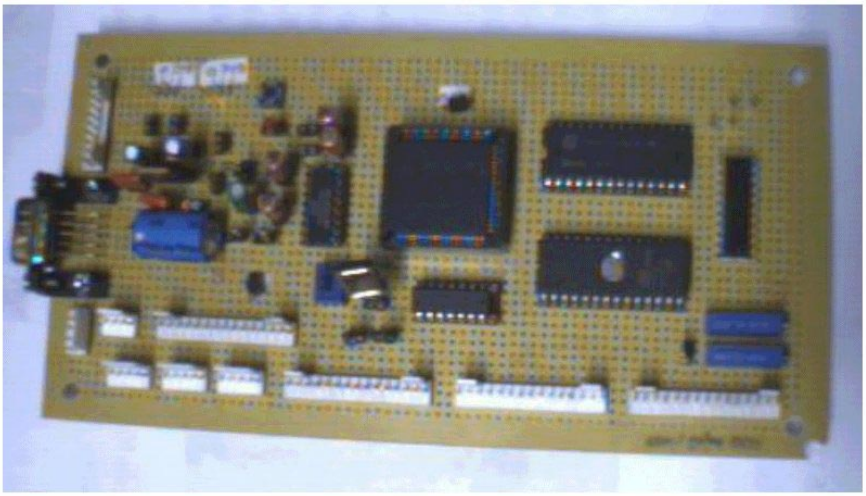


Figura 22 - Placa principal feita em wire-wrap. **Créditos: Fr&d**

5.4.5 Sensores

Para maximizar o efeito dos sensores de colisão implementamos uma espécie de pára-choque frontal. A frente do robô conta com dois desses interruptores de colisão, um do lado esquerdo e outro do lado direito. Cada uma das extremidades do pára-choque é afixada em um dos interruptores. Se o objeto com o qual o robô colidiu estiver no centro os dois interruptores serão acionados. Caso o objeto esteja em um dos lados, somente um dos interruptores será pressionado, indicando o lado da colisão.

Com relação aos sensores de luminosidade (*LDR*), decidimos por deixá-los sozinhos, isto é, conectados ao circuito apenas por um cabo com conector molex, deixando os trim-pots (que fazem parte do divisor de tensão *LDR*/resistor) afixados na placa principal. Isto proporciona maior mobilidade aos *LDRs* que podem ter sua posição e direção alteradas, dependendo do comportamento desejado no robô. Os valores dos *trimpots* (10Kohm) foram especificados com base nos valores de resistência dos LDRs: 40Kohm para escuro e praticamente zero para sol forte.

A montagem dos sensores de distância *IRPD* e do *Tracker* são muito simples bastando soldar os componentes na placa. Porém há a necessidade de ajustes para o funcionamento. Percorre-se o potenciômetro de sensibilidade até que esta esteja de acordo com o desejado. Um recurso que é descrito no manual e que pode melhorar o desempenho do detetor de infravermelho é colar várias camadas de fita adesiva transparente (*Durex™)* na frente. Isto reduz um pouco a sensibilidade do detetor, mas melhora seu comportamento. A perda de sensibilidade pode ainda se compensada por meio do potenciômetro.

6. Descrição do Software do Computador

O s*oftware utiliz*ado no computador para criar os comportamentos do robô mantém a estrutura especificada. Temos toda a operação centrada num editor de comportamentos, responsável fornecer recursos de auxílio à edição do código. A partir do editor, o usuário poderá ainda compilar o comportamento escrito em linguagem procedural ou em *subsumption*. Por fim, é ainda possível enviar o comportamento diretamente ao robô.

6.1 Descrição Funcional e Estrutura

Como veremos adiante, decidimos utilizar dois *aplicativos* já existentes comercialmente ao invés de desenvolvê-los. Assim optamos para focalizar nossos esforços no verdadeiro escopo do projeto, que é a apresentação de uma arquitetura para robô e a sua programação através de *subsumption*. Assim, caso haja o interesse de alguém montar o "laboratório de *subsumption*" por nós proposto, ele terá que adquirir ambos os *softwares* ou adaptar equivalentes.

Outro ponto que pode causar polêmica é que, no compilador de *subsumption* que criamos, toda a interface e os comentários no código estão em *inglês*. Nossa motivação principal foi o intuito de disponibilizá-lo na *Internet* para que vários programadores e usuários contribuam com sugestões. Como esperamos atingir grupos fora do país, nada mais natural do que utilizar o idioma universal da *Internet*. Cremos que seria difícil a manutenção e sincronização de dois aplicativos. De qualquer modo, esperamos modificar o compilador para basear toda as suas mensagens de interface num arquito texto que pode ser modificado e traduzido para qualquer idioma.

Por fim, um ponto relativamente óbvio, mas que vale a justificativa. Todo o *software* do computador foi feito para a plataforma *Microsoft Windows*. Apesar da grande popularidade do *Linux* (e de nossa simpatia e preferência por ele), decidimos criar aplicativos que atendessem a maior parcela possível de usuários. Futuramente, esperamos portar o programa para várias plataformas, certamente utilizando *Java*.

6.1.1 A linguagem SOUL

O nome *SOUL* vem das iniciais de *Subsumption Oriented Uniform Language*, que caracteriza a linguagem como sendo orientada ao modelo de *subsumption.* Trata-se basicamente de uma variação da linguagem de descrição de comportamentos *subsumption* apresentada informalmente em [Jones&Flynn93].

Descreve os diagramas de processos *subsumption* de forma textual, detalhando todo o comportamento e suas conexões internas. Apresenta uma organização e estrutura de dados próprias para permitir que o usuário crie comportamentos completo sem a necessidade de artifícios que poderiam complicar o entendimento ou comprometer a estabilidade lógica do robô.

Mantivemos a estrutura básica de [Jones&Flynn93], introduzindo códigos em C internamente a cada estado dos processos do comportamento do robô. A vantagem é que podemos ter códigos mais poderosos associados a cada estado. Porém, como desvantagem, temos a dependência em relação ao C e uma menor portabilidade entre microcontroladores. Outra alteração foi na nomenclatura. Chamamos de *process* (processo) onde eles chamam de *behavior*, e *behavior* (comportamento) onde aparece *behavior network.*

Apesar de possuir o mesmo objetivo, a linguagem *SOUL* é bastante diferente da *Behavior Language* apresentada por Rodney Brooks em [Brooks90a]. Lembrando que Brooks foi quem criou o modelo de *subsumption* [Brooks90b] [Brooks86] [Brooks89]. A *Behavior Language* utiliza uma estrutura bastante detalhada e poderosa para descrever comportamentos *subsumption*, assim como o conceito de *hormones* responsáveis por mudar as prioridades dos comportamentos sob certas condições. Uma listagem exemplo de um comportamento em *SOUL* pode ser visto no *Anexo VI*.

6.1.2 Interface

O editor de código de comportamento não tem restrições. Pode ser qualquer editor que suporte texto ASCII. Os compiladores podem ser ativados manualmente, mas alguns editores possuem recursos para integração com outros aplicativos, facilitando bastante o processo edição-compilação.

Porém, nossa escolha para esse projeto e recomendação é pelo *UltraEdit-32 Professional Text/HEX Editor Version 6.20b* da empresa americana *IDM Computer Solutions*. Trata-se de um excelente editor de códigos que é distribuído na forma de *shareware* na *Internet* a um preço de US$30,00.

Ele aceita textos em ASCII ou EBCDIC, ANSI ou OEM, e DOS ou UNIX ou MAC. Permite que o usuário defina uma descrição da linguagem que irá editar, definindo vários níveis de palavras-chaves (*keyword*), formas de comentários, formato de funções (para uma lista delas em tempo de edição), além de várias conversões no formato do texto, macros e itens de menu customizados para certas ações, como executar outro aplicativo.

Para integrar algum aplicativo ao *UltraEdit-32*, basta criar um novo item de menu, definir que a ação é executar um programa e preencher com "`c:\comp\compiler %F`", onde `compiler` é o nome do compilador e `%F` é substituído pelo caminho completo da janela sendo editada.

A descrição da linguagem *SOUL* para tal programa está no *Anexo VII*.

6.1.3 Compilação Procedural

A compilação de comportamentos procedurais é a compilação de um programa em C. Trata-se de um nível razoável entre poder de programação (com o uso de estruturas de dados), acesso aos recursos do microcontrolador (versus o que se faz com *Assembly*) e familiariade do usuário.

O ambiente *Keil PK51 2K* que recebemos como doação da *Anacom Software* foi bastante utilizado para a criação de várias rotinas de acesso aos sensores. Suas ferramentas de debugação e simulação são poderosas o suficiente para que a maioria dos problemas fosse resolvida acompanhando-se os estados dos registradores e valores de memória.

Porém, não utilizamos tal ambiente no projeto final. um motivo é o limite de programas de 2Kbytes. Apesar de a empresa ter se disponibilizado em compilar programas acima desse valor caso fosse necessário, dispunhamos de um outro compilador sem essa limitação. Alguns de nossos comportamentos de testes já estavam ultrapassando tal limite e precisávamos de maior agilidade.

Além disso, caso o projeto seja utilizado mais tarde por outros grupos no Laboratório de Projeto de Formatura ou de Microprocessadores nos anos seguintes, precisávamos de plataforma mais acessível.

Nossa opção foi pelo *AvCase51* da empresa *Avocet*. Apesar de não ter os recursos disponíveis nos ambientes da *Keil*, o grupo já tinha familiariade devido a disciplina em semestres anteriores.

Como se trata de um *software* comercial, o usuário que desejar utilizá-lo, e que não seja aluno dos laboratórios citados, deve adquirí-lo ou adaptar para uma solução similar.

6.1.4 Compilação *Subsumption*

Um compilador de comportamentos em *SOUL* para linguagem C foi desenvolvido. O nome dado foi *ESCompiler* ou apenas *ESC* (que significa as iniciais de *ESC is a Subsumption Compiler*). Ele toma um comportamento feito pelo usuário e um *driver* para o microcontrolador alvo, e fornece um código em C que pode ser enviado para o *Compilador Procedural* descrito.

6.1.4.1 Exemplo

Dado o comportamento em *SOUL* listado em *6.1.1 - A linguagem SOUL*, adicionamos o *driver* correspondente para o P80C552 e o resultado da compilação pode ser checado *Anexo VIII*. Observação: estamos ciente que listagens

longas de programas não são adequadas para tal tipo de documentação, principalmente quando existe a disponibilidade dos meios digitais; pedimos tolerância.

6.1.4.2 Interface

A interface é simples. Possui campos para os arquivos de entrada em SOUL (*source*), saída em C (*target*), registro de atividades e mensagens (*log*), e o *driver* do microcontrolador (*driver*). As opções que foram implementadas são: inserção de comentários do compilador, criar ou não um arquivo de registro (*log*) e fechar o programa ao terminar a compilação.

Os arquivos podem ser abertos por botões específicos (com "...") , com *drag-and-drop* utilizando o *mouse* e por passagem de parâmetros ao executar o compilador.

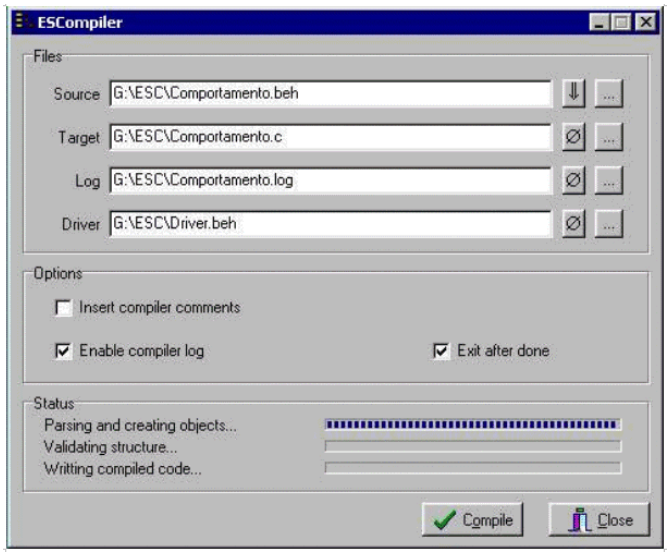


Figura 23 - Interface do compilador subsumption (ESCompiler). **Créditos: Fr&d**

6.2 Estruturas de Dados Utilizadas

6.2.1 Diagrama de Objetos

Foram criados diversos objetos na codificação do *ESCompiler* para tratar de cada uma das ações.

Apenas os listamos as classes que o definem a seguir:

- TIDE: É a tela de interface com o usuário. Chama os objetos que realizam a compilação, passando para eles as informações entradas pelo usuário.
- TBrain: Coordena toda a compilação, criando os objetos que executam as sub-operações e chamando-os de acordo com a lógica da compilação.
- TFileIO: Responsável por toda a interface com os arquivos. Possui chamadas para carregá-los em objetos de memória do compilador e disponibilizam métodos para que os outros objetos escrevam mais facilmente nos arquivos de registro e saída.
- TCodeReader: Lê o código fonte, separando o texto em palavras e *tokens* significativos e enviando-os para um autômato determinístico. Esse autômato também foi inteiramente codificado, ao invés de utilizarmos ferramentas

disponíveis como o *Bison* [Miano97], para permitir maior integração com o resto do *software*, além de ser outro tópico de interesse que o grupo decidiu explorar. O autômato possui todas as rotinas semânticas para criar uma estrutura de objetos que representa o código fonte num formato mais acessível para o resto dos objetos de transformação.

- TValidator: Realiza uma checagem e validação da estrutura de objetos. Avalia sobre a existência de nomes de processos e entradas repetidos, atribuição de valores a entradas de processos, e referência a estados não existentes.

- TCodeWriter: Transforma a estrutura de objetos em linguagem C, criando a toda a comunicação entre processos, o código main, e a inserção do *driver* para que o resultado seja um único código em linguagem em C que possa ser compilado pelo *AvCase*.

- TProcess: Comporta um processo completo, com objetos internos para armazenar suas características, códigos em C antes e depois dos estados e cada um dos estados.

- TESCParser: Separa as linhas do código em palavras e *tokens*, informando ainda se uma palavra é uma palavra chave (*keyword*).

- TESConnection: Representa uma conexão entre processos.

- ESCeption: Uma exceção especial para tratar de erros do compilador.

6.3 Recursos de Desenvolvimento

Para a criação do *compilador subsumption ESCompiler* (*6.1.4 - Compilação Subsumption*) e o programa de testes do braço articulado (*8 - Outros Softwares*), utilizamos o ambiente *Borland C++Builder 3.0*. Tal escolha foi basicamente pelo fato de ser a única linguagem de programação disponível no laboratório.

Foi uma escolha com o aval do grupo inteiro, pois as outras opções que poderiam ter sido utilizadas tinham alguns problemas: *Microsoft Visual Basic* por sua lentidão, falta de recursos e propensão a *bugs*; *Java* (o ideal) porque o computador de desenvolvimento no laboratório não suportaria ambientes como o *Borland JBuilder*; e *Borland Delphi* porque nem todos no grupo estavam familiarizados com *Object Pascal*.

7. Descrição do Software do Robô

O software do robô tem como finalidade última executar o comportamento que o usuário desenvolveu. Como a proposta é um sistema onde pode-se testar vários comportamentos, é recomendado utilizar algum esquema onde se possa mudar o código com facilidade diversas vezes. Este objetivo pode ser satisfeito de diversas maneiras, por exemplo *EEPROM* e memórias *flash.* O esquema adotado no nosso projeto foi igual ao utilizado nos *kits* da disciplina *PCS598 – Laboratório de Microprocessadores*, onde é possível colocar código numa área de RAM que é mapeada para o microcontrolador como se fosse memória RAM. Este esquema é mais simples e barato, pois dispensa gravadores especiais e permite usar microcontroladores de custo um pouco mais baixo.

Um dos requisitos para se utilizar tal esquema é um programa que consiga estabelecer comunicação com o PC, copiar o código executável de *80C51* e transferir o controle para a posição de memória onde este código será copiado. No protótipo foi utilizado o *Programa Monitor*, desenvolvido pelos docentes da disciplina PCS598. Foram necessárias modificações no programa monitor para se adequar a um clock de 16MHz e às interrupções especiais presentes no microcontrolador utilizado no robô, o *80C552*, da *Philips*.

Além do *Programa Monitor*, também roda no microcontrolador um programa que permite que o robô execute os comportamentos criados pelo usuário. Este programa é o *software* do robô.

7.1 Programa monitor modificado

Para que pudéssemos utilizar um *clock* de 16MHz tivemos que alterar o valor carregado no registrador *TH1* pelo programa monitor. O registrador *TH1* faz parte de um *timer* cujo *overflow* determina a baud rate do canal serial do microcontrolador. Como a taxa de contagem do *timer* varia com a freqüência de relógio do microcontrolador, a taxa de transmissão também varia.

A primeira modificação foi, então, substituir o valor de inicialização de *TH1*. O valor anterior, FDh, que gera uma *baud rate* de 9600 bps a 11.0592MHz foi substituído por EFh que gera uma taxa de 2400bps a 16.000MHz.

A outra modificação foi a inclusão de suporte às interrupções especiais da família *8XC552*. A *Tabela 3* apresenta tal modificação.

Tabela 3 - Endereços das Interrupções

Endereço original	Descrição	Com jump para
002Bh	Interrupção da bus I2C	FFD0h
0033h	Interrupção 0 de capture do timer 2	FFD3h
003Bh	Interrupção 1 de capture do timer 2	FFD6h
0043h	Interrupção 2 de capture do timer 2	FFD9h
004Bh	Interrupção 3 de capture do timer 2	FFDCh
0053h	Interrupção de final de conversão A/D	FFDFh
005Bh	Interrupção 0 de compare do timer	FFE3h
0063h	Interrupção 1 de compare do timer	FFE6h
006Bh	Interrupção 2 de compare do timer	FFE9h
0073h	Interrupção 3 de compare do timer	FFECh

Os endereços originais de *jump* para a área de código mapeada em *RAM* do programa monitor permanecem inalteradas.

7.2 Escalonador, processos de usuário e drivers

Toda vez que o usuário desejar fazer o *download* de comportamentos para o microcontrolador, este será inserido num código-fonte e compilado. Este código fonte contém a especificação do comportamento recém-criado pelo usuário, e também drivers para acesso a sensores e atuadores, e um esquema de escalonamento para rodá-los de maneira conveniente. Maior detalhamento na seção abaixo.

7.3 Descrição Funcional e Estrutura

O software é organizada na forma de um *loop infinito***,** que executa as seguintes tarefas:

- Checa a entrada de todos os sensores
- Roda os processos de *subsumption*, computando as saídas de cada um
- Baseado na prioridade destes processos, envia as saídas dos processos aos atuadoes (rodas, braço e buzzer).

Cria-se assim, um esquema de *multitarefa cooperativo*, onde dependemos de cada processo retornar o controle para que o próximo assuma e a execução continue normalmente.

7.4 Estruturas de Dados Utilizadas

Cada processo *subsumption* de usuário acaba sendo compilado como um conjunto de variáveis globais, representando suas entradas e saídas, que são a interface com o resto do programa. O processo propriamente dito (isto é, o algoritmo) gera uma função com o nome do processo. A geração das variáveis globais e da função é papel do *ESCompiler.*

```
byte processo_InfravermelhoEsq;
byte processo_InfravermelhoDir
byte processo_MotorEsq;
byte processo_MotorDir;
byte processo_Step;
void processo() {
    if (processo_InfravermelhoEsq == 1) {
        processo_MotorEsq = FAST;
        processo_MotorDir = SLOW;
    }
}
```

No exemplo acima, *processo_InfraVermelhoDir* e *processo_InfraVermelhoEsq* representam as entradas do processo, enquanto *processo_MotorEsq* e *processo_MotorDir* representam as saídas do processo. O exemplo de processo desvia de alguma coisa ao perceber obstáculos sendo detetados pelo sensor infravermelho esquerdo (faz isso virando para a esquerda, ao girar a roda esquerda mais que a direita). Assim como este processo exemplo, cada processo gerado pelo *ESCompiler* tem entrada e saídas precedidas pelo próprio nome. Fica a cargo de um processo do sistema alimentar estas variáveis com as entrada dos sensores e enviar o comando do processo mais prioritário que esteja rodando para as saídas. As entradas dos sensores ficam guardadas em variáveis globais também, mas sem nenhum nome de processo precedendo-as (assim a “verdadeira” entrada dos infravermelhos fica guardada numa variável chamada *InfravermelhoDir* ou *InfravermelhoEsq*).

É definida ainda uma variável chamada *processo_Step*, que determina de quanto em quanto tempo é rodado o processo em questão. Cada processo tem uma variável *Step* associado. Maiores informações sobre geração de processos na descrição do *ESCompiler*, neste mesmo documento.

7.5 Recursos de Desenvolvimento

Para o desenvolvimento dos *drivers* de acesso aos sensores e dos programas do robô foram utilizados dois compiladores de linguagem “C” para o *80C51*: *Keil PK-51* e *Avocet AvCase51*.

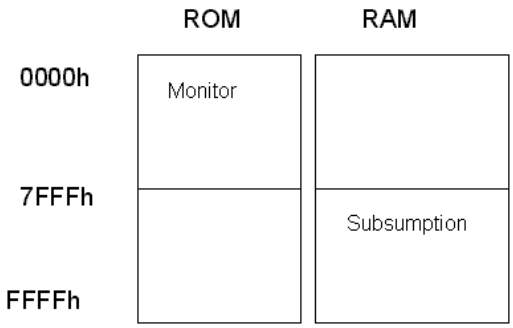
As ferramentas da *Keil* são extremamente poderosas e contam com recursos para debugação ideias para profissionais da área. Elas foram de grande importância para que o grupo criasse a arquitetura de software no robô. Porém, como se trata de uma ferramenta comercial, resolvemos por não utilizá-la na implementação final, simplesmente porque iria contrariar nosso intuito de tornar a projeto o mais acessível possível.

Assim, por se tratar da ferramenta adotada pelo Laboratório, adotamos o *AvCase 51*. Os comportamentos *subsumption* criados pelo usuário foram todos compilados nele. Destacamos a facilidade com que ele pode ser chamado da linha de comando e a grande possibilidade de customização e controle sobre a compilação.

7.6 Mapa de Geração de Código Executável/ Gravação de Memória

O programa que executa o modelo de *subsumption* é gerado a partir do arquivo *Fred.c*, que por sua vez é gerado pelo *ESCompiler.* O *AvCase* é chamado automaticamente para compilar o *Fred.c*, e o produto final é um arquivo chamado *Fred.hex*, que é enviado pelo *WinTalk* para o robô.

O diagrama abaixo mostra como se distribuem os programas na memória



8. Outros Softwares

8.1 Necessidade

Com a conclusão da montagem do braço mecânico e da base do robô, foi possível testar o funcionamento deles antes da implementação da placa de controle com o microcontrador da família 80C51. Para nos familiarizarmos com o modo de comunicação com o braço mecânico, utilizamos o próprio computador da bancada, através da porta serial COM2. Assim, evita-se problemas de hardware durante essa fase inicial que envolve apenas comandos lógicos.

O *software* para testes que acompanha o braço mecânico utiliza um compilador BASIC STAMP, ainda em MS-DOS, para efetuar testes com o braço. Cientes de que necessitaríamos maior controle, decidimos desenvolver um programa dedicado a isso. Além disso, era objetivo controlar os servos através do *mouse*, possibilitando ajustes finos e empíricos (ao contrário do cálculo de posições), além de uma análise da resposta a variações à movimentos naturais.

Um programa foi desenvolvido em *Borland C++ Builder 3.0* [Calvert97] [Miano97] [Reisdorph98], possibilitando ao usuário controlar por *mouse* e por posição absoluta cada servo, além de possibilitar uma programação rudimentar.

8.2 Configuração

Inicialmente, o programa apresenta uma tela de configuração, como pode ser visto na *Figura 24.* Nessa tela, é possível configurar o nome de cada servo e as posições limites de cada um. Isso é muito importante para evitar que o usuário force alguma estrutura que tem percurso limitado devido à construção. Em nosso robô, o caso mais crítico é a garra. Além disso, pode-se configurar qual a porta de comunicação utilizada e qual a taxa de comunicação (2400 ou 9600 baud). Todos os dados são armazenados em arquivo e automaticamente lidos e gravados.

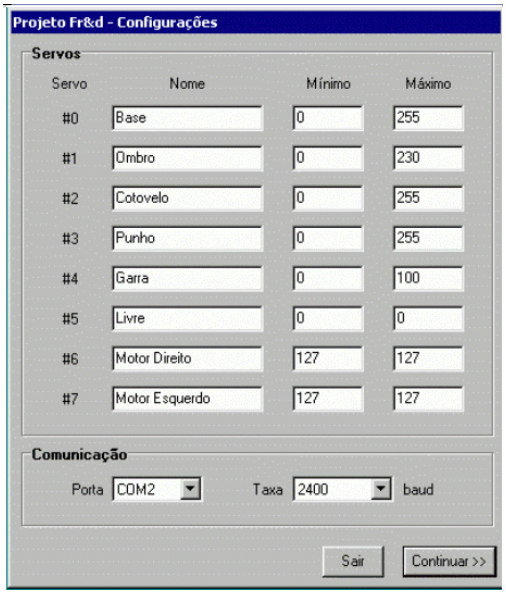


Figura 24 - Configuração do Programa. **Créditos: Fr&d**

8.3 Operação

A tela de operação do braço pode ser vista na *Figura 25.* Existem oito barras de rolagem para cada um dos canais do controlador do robô. No caso, são 5 para o controle do braço, dois para os motores de locomoção e 1 livre. Essas barras podem ser movidas com o *mouse*, sendo que o braço responde imediatamente. Na verdade, apenas a saída PWM do controlador do robô é sincronizada e em tempo real com o programa. A posição do braço depende da

resposta do servo-mecanismo, mas é particamente instantânea do ponto de vista do usuário. Também a possibilidade de se alterar numericamente cada um dos servos, indicando qual servo e qual a posição.

O espaço denominado *Bookmarks* é onde várias posições do braço podems ser armazenadas, criando assim um atalho para cada uma e até mesmo uma programação. Basta acertar nas barras de rolagem uma posição e adicioná-la nos *Bookmarks* através do botão direito do *mouse*. A nova posição pode ser adicionada ao final da lista ou inserida em qualquer posição. Esses atalhos podems ser ainda um a um apagados, ou a lista inteira é limpada. Para que o braço vá a uma posição gravada, basta clicar no nome correspondente. O braço se move de acordo com as configurações que veremos abaixo, interpolando de forma linear as duas posições.

Existe ainda a opção de colocar em *Automático* com o botão direito do *mouse,* onde temos as várias posições sucessivas sendo chamadas e aplicadas ao braço. Roda-se então um *script* composto pelas várias posições, porém sem nenhum teste, laço ou espera entre os movimentos. Por fim, o *Modo Contínuo* coloca o uma repetição contínua do modo *Automático*.

Por último, é possível configurar qual o intervalo em milissegundos entre cada atualização do servo, assim como quantos passos são incrementados por vez. Algumas observações. Existem 255 passos ou posições para cada servo. O valor 255 pode ser configurado para valer 90° ou 180°. Para evitarmos confusões ou números “quebrados”, preferimos manter as posições inteiras. Além disso, o intervalo entre cada atualização não significa que cada posição é atualizada a cada *n* milissegundos, apenas diz o intervalo entre elas. Cada atualização envia o comando pela serial, espera o fim da transmissão e então termina.

Vários valores de passo e intervalo foram testados. Achamos porém que passos muito grandes levam o braço a vibrações indesejadas. Estamos preparando um estudo mais detalhado sobre isso, porém dentro do escopo de nossos objetivos. Também existe o cuidado de não realizar movimentos em muitos eixos entre uma posição gravada e outra, pois diminui a taxa de atualização em cada eixo em separado e levando a oscilações.

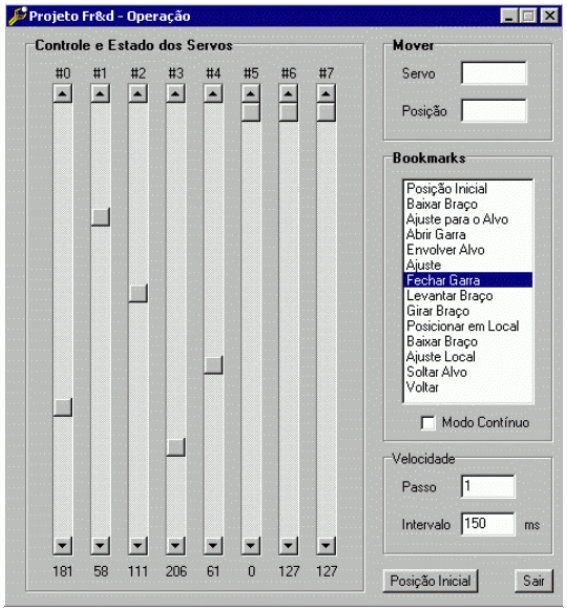


Figura 25 - Operação do Programa. **Créditos: Fr&d**

9. Avaliação e Testes do Projeto

Até a data de redação deste documento, não havíamos ainda testado o robô em pleno funcionamento. Apenas alguns testes iniciais com componentes isolados foram feitos.

9.1 Recursos de Apoio Utilizados

Utilizamos apenas programação em *C++Builder* no computador e um cabo serial.

9.2 Testes Efetuados

9.2.1 Operação do braço articulado

Como descrevemos no item *8 - Outros Softwares*, realizamos testes com o braço articulado enviando pacotes de comando através do computador. Mesmo quando enviávamos sucessivos pacotes, o controlador *Mini SSC II* se comportou como esperado. Esses testes foram importantes para avaliarmos a resposta do braço e sua precisão de movimentos.

9.2.2 Sensores de proximidade

Com a placa principal de controle do robô em operação, criamos um programa para testar o estado dos sensores de proximidade e enviar comandos pela serial para o computador. Pudemos observar dinamicamente qual a resposta dos sensores para diferentes tipos de objetos e à diferentes distâncias.

10. Cronograma de Desenvolvimento Efetivamente Cumprido

Tabela 4 - Cronograma

	JUL (1 a 17)	JUL(17 a 31)	AGO (1 a 14)	AGO (15 a 31)	SET (1 a 18)	SET (19 a 30)	OUT (1 a 16)	OUT (17 a 31)	NOV (1 a 13)	NOV (14 a 30)	DEZ (1 a 10)	DEZ (11 a 16)
Pesquisa de Hardware												
Pesquisa de Software												
Estudo da Teoria												
Compras												
Operação dos Sensores												
Operação do Microcontrolador												
Interligação dos Sensores												
Software Básico												
Rotinas dos Sensores												
Compilador Procedural												
Compilador Subsumption												
Integração do Software												
Montagem dos Componentes												
Testes												
Documentação												

11. Empacotamento

Decidimos por deixar o circuito sem nenhuma caixa ou algo parecido. A placa do circuito principal fica afixada diretamente à base do robô. A largura da placa é exatamente a mesma do robô, de modo que ela fica protegida pelo corpo do robô. Entendemos que a placa com os componentes aparentes também exerce função estética, bem como as placas dos sensores. Por isso, a idéia foi deixar tudo à mostra.

12 Manual do Usuário

12.1 Projeto Fr&d

12.1.1 Componentes do Hardware

O robô montado tem o aspecto da *Figura 26*.

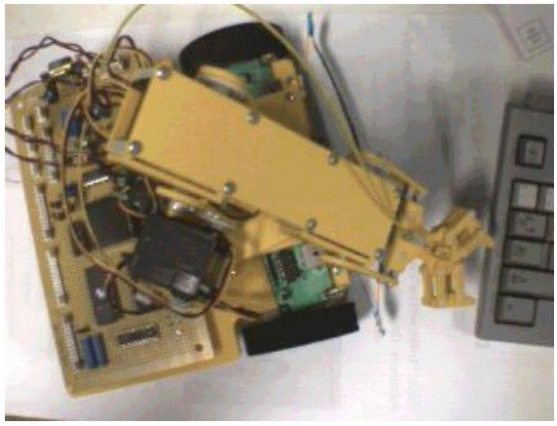


Figura 26 - Robô ao lado de teclado. **Créditos: Fr&d**

A lista abaixo descreve as partes constituintes do hardware do robô:

- Base robótica – Constituída do suporte plástico, dos servomecanismos e das rodas. Serve de base para os sensores e a placa principal.

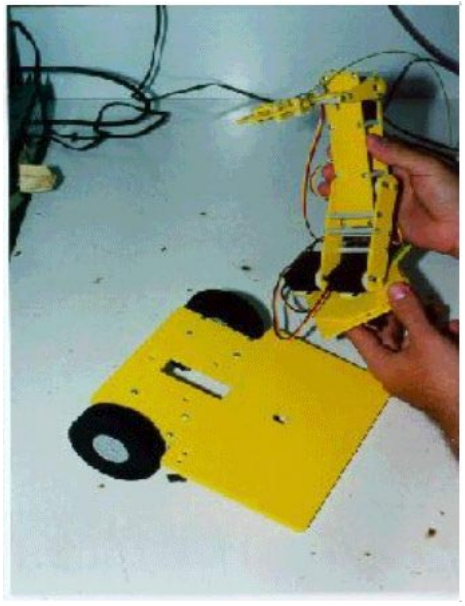


Figura 27 - Base robótica. **Créditos: Fr&d**

- Placa principal – Realiza todo o processamento do robô. Vai montada em cima da placa principal. É nela que é feita a ponte entre sensores e motores.

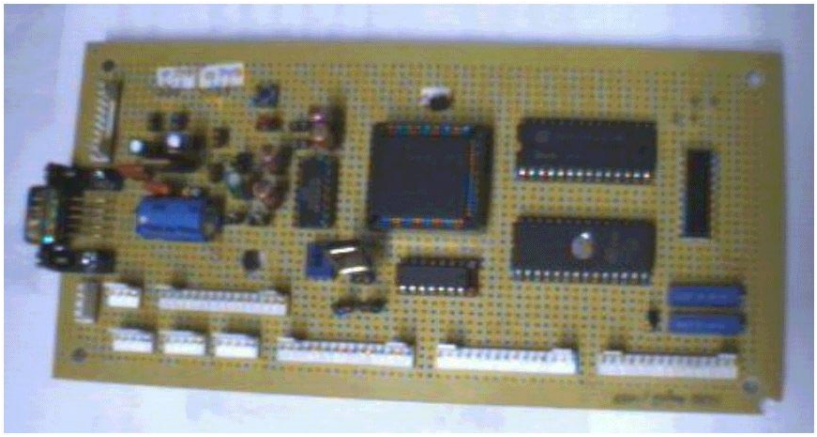


Figura 28 - Placa principal. **Créditos: Fr&d**

- MINI SSCII – (*Mini Serial Servo Controller)* – Um produto da Lynxmotion[*Lynxmotion*] que faz parte do robô. É utilizado para controlar os oito servomecanismos existentes no robô.

Figura 29 - O Mini SSC corresponde aos circuitos na base do braço. **Créditos: Fr&d**

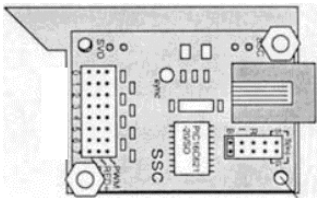


Figura 30 - Desenho do Mini SSC. **Créditos: [Lynxmotion98a]**

- Baterias – A sugestão de uso é de 5 pilhas do tipo C (média) e de uma bateria de 9V para o Mini SSC. As 5 pilhas do tipo C servem para alimentar os servomecanismos, a placa principal e os sensores, a bateria de 9V serve para alimentar o Mini SSC que, segundo o fabricante, não funciona se ligado à mesma fonte que os servomecanismos. Quanto a requisitos de corrente/tensão, para alimentar a placa principal, motores e sensores, são necessários 800mAh, e uma tensão de 7V a 8V. Para alimentar o mini SSC é necessária uma tensão de 7 a 15V e uma corrente da ordem de 150mAh.

- Sensores – Os sensores que acompanham o robô encontram-se na forma de placas adicionais (infravermelhos e seguidor de trilha), ou então na forma de pequenos dispositivos na ponta de cabos, como é o caso do microfone e dos foto-resistores.

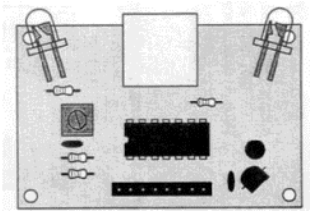


Figura 31 - Ilustração do sensor de deteção infravermelho. **Créditos: [TSS97]**

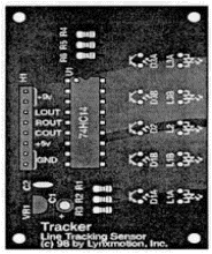


Figura 32 - Aspecto do circuito seguidor de trilhas. **Créditos: [Lynxmotion98d]**

- Cabo serial – Para conectar o PC ao robô e fazer o *download* de comportamentos. O cabo usado é do mesmo tipo com que se faz comunicação serial entre dois computadores pessoais ou entre um computador pessoal e um *switch* ou roteador. Numa das pontas é preciso ter um conector DB9 fêmea, porque a placa principal se co7necta através de um DB9 macho.

- Conectores para bateria e sensores – São usados para ligar sensores e bateria ao robô conectores do tipo Molex. Cada cabo tem um identificação de onde deve ser ligado.

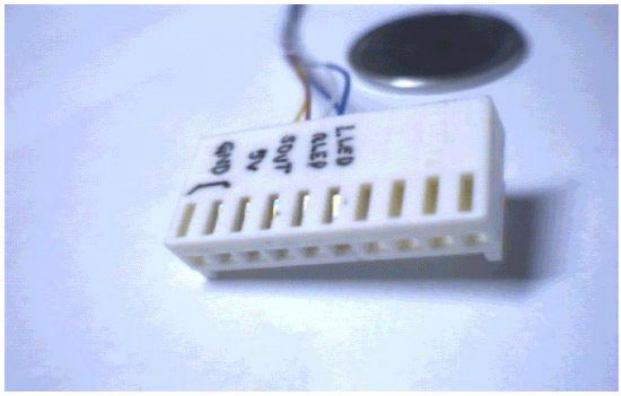


Figura 33 - Detalhe de conector Molex. **Créditos: Fr&d**

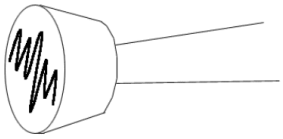


Figura 34 - Foto-resistor. **Créditos: Fr&d**

12.1.2 Configuração e montagem

12.1.2.1 Ligação de conectores e sensores

A placa principal tem 9 portas com conectores Molex macho, servem para conectar os diversos tipos de sensores.

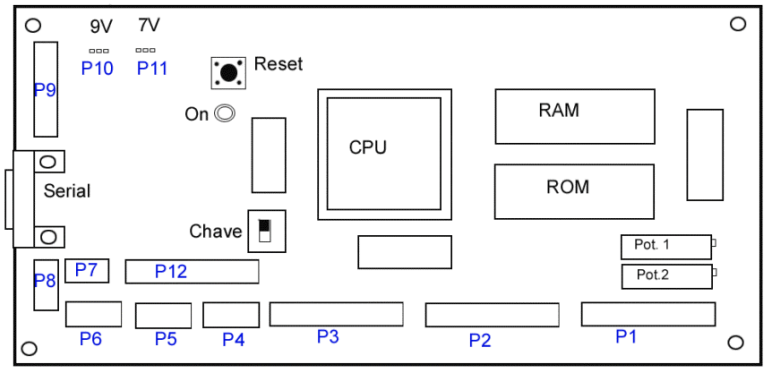


Figura 35 - Placa principal de controle. **Créditos: Fr&d**

As portas identificadas com os números 1 e 2 no desenho servem para conexão dos foto-resistores e do microfone. Os cabos dos foto-resistores devem ter o próprio foto-resistor numa das pontas e um conector *Molex* fêmea de três pinos numa outra ponta.

Os conectores *Molex* dos foto-resistores devem ser colocados nas portas 1 e 2, de acordo com a numeração do desenho. É importante observar o espaçamento de 1 pino entre conectores (vide desenho abaixo). As portas 1 e 2 estão conectadas aos conversores A/D do microcontrolador do robô.

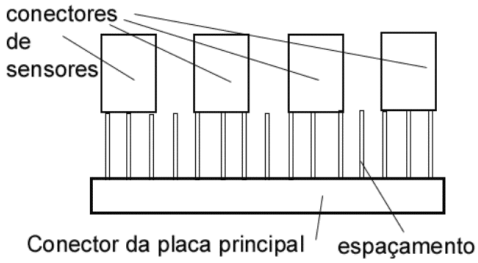


Figura 36 - Espaçamento entre os pinos. **Créditos: Fr&d**

Os quatro sensores de deteção infravermelhos devem ser conectados pelos cabos que têm um conector Molex fêmea de 5 pinos numa ponta e um de 10 pinos na outra ponta (o mesmo da foto do item sensores). Os sensores infravermelhos estao distribuídos no robô conforme o desenho abaixo.

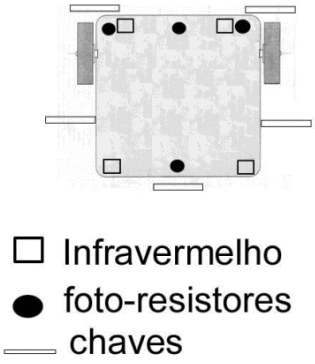


Figura 37 - Posição dos sensores. **Créditos: [Lynxmotion98b] e Fr&d**

Os sensores infravermelhos devem ser conectados às portas 4, 5, 6 e 8 da placa principal. O seguidor de trilha deve ser conectado à porta 7.

Além dos sensores já mencionados, também podem ser ligadas as chaves de contato, que são 5 usadas no robô. As chaves de contato ligam-se através de conectores *Molex*, e são ligadas na porta 3. A disposição das chaves de contato é: 2 na frente (suporte do pára-choque), uma em cada lateral e 1 atrás do robô.

A *Tabela 5* abaixo mostra as conexões que podem ser alteradas pelo usuário relativas a sensoreamento.

Tabela 5 - Conexão dos sensores na placa principal

Sensor	Conector	Posição no robô	Tipo
Sensor de deteção infravermelho 01	8	Dianteira esquerda	Entrada nível TTL
Sensor de deteção infravermelho 02	4	Dianteira direita	Entrada nível TTL
Sensor de deteção infravermelho 03	5	Traseira esquerda	Entrada nível TTL
Sensor de deteção infravermelho 04	6	Traseira direita	Entrada nível TTL
Chave de contato 1	3	Dianteira esquerda	Entrada nível TTL
Chave de contato 2	3	Dianteira direita	Entrada nível TTL
Chave de contato 3	3	Lateral esquerda	Entrada nível TTL
Chave de contato 4	3	Lateral direita	Entrada nível TTL
Chave de contato 5	3	Traseira	Entrada nível TTL
Sensor seguidor de trilha	7	Inferior (embaixo da base)	Entrada nível TTL
Foto-resistor 1	1	Dianteira esquerda	Conversor AD
Foto-resistor 2	1	Dianteira central	Conversor AD
Foto-resistor 3	1	Dianteira direita	Conversor AD
Foto-resistor 4	1	Braço mecânico	Conversor AD
Foto-resistor 5	2	Traseira	Conversor AD
Microfone	2	Superior	Conversor AD

Ambas as alimentações (para placa/sensores/motores e para Mini SSC) devem ser conectadas em conectores Molex da placa principal. A alimentação vinda das pilhas do tipo C (principal) deve ser conectada na porta 11, e a alimentação para o Mini SSC deve ser conectada na porta 10.

Tabela 6 - Alimentação dos circuitos

Alimentação	Para	Conector da placa principal
5 pilhas do tipo “C”, 7V a 8V, 8000mAh	Placa, sensores e motores	11
9V	Mini-SSC	10

Por último, é necessário conectar a base robótica à placa principal. Para isso deve-se conectar o cabo que vem da base robótica no conector 9 da placa principal.

12.1.2.2 Mapeamento dos sensores nas portas do microcontrolador

O microcontrolador utilizado, como já foi mencionado, é um 80C552. Este microcontrolador é da mesma família que o 80C51 da *Intel* e compatível com este. Ele tem as seguintes características:

- Três *timers*/contadores
- Quatro registradores de captura das entradas e três registradores de comparação das entradas
- Memória interna de 384 bytes
- Conversor A/D de 10 bits multiplexado por oito pinos
- Duas saídas PWM
- Cinco portas de entrada/saída digitais e uma porta para conversão A/D
- UART compatível com a do 80C51
- Clock de 16MHz

A *Tabela 7* mostra como as portas do 80C51 estão mapeadas nos periféricos (não confudir as portas do microcontrolador com as portas do ítem *12.1.2.1 Ligação de conectores e sensores***,** que são apenas conectores de onde partem os cabos. As 5 portas comuns do microcontrolador são identificadas como P0, P1, P2, P3 e P4. A porta com conversão A/D é chamada de P5. Dentro de uma mesma porta os pinos (que são oito) são identificadas como P*n*.0, P*n*.1, P*n*.2, P*n*.3, P*n*.4 e P*n*.5. O índice *n* indica o número da porta.

Tabela 7 - Mapeamento dos sensores por porta

Pino 80C552	**Periférico**	**Comentários**
P0	Interface com memória (via de dados/endereço)	P0 é usada para comunicação entre 80C51 e memórias
P1.0	LED emissor direito dos detetores infravermelhos	Uma mesma porta comanda os 4 LEDs dos infravermelhos
P1.1	Led emissor esquerdo do detetores infravermelhos	Simétrico ao anterior
P1.2	Receptor do detetor infravermelho 1	A recepção de cada detetor é tratada separadamente. Detetor dianteiro esquerdo
P1.3	Receptor do detetor infravermelho 2	Detetor dianteiro direito
P1.4	Receptor do detetor infravermelho 3	Detetor traseiro esquerdo
P1.5	Receptor do detetor infravermelho 4	Detetor traseiro direito
P2	Interfaceamento com memória (via de endereços)	
P3.0	Pino de recepção para comunicação serial	
P3.1	Pino de transmissão para comunicação serial	
P3.3	Pino para interrupção externa	Pode ser usado para tarefas sensíveis a atraso, como espera pelo pulso de retorno de um sonar
P4.0	Chave de contato 1	Chave dianteira esquerda
P4.1	Chave de contato 2	Chave dianteira direita
P4.2	Chave de contato 3	Chave lateral esquerda
P4.3	Chave de contato 4	Chave lateral direita
P4.4	Chave de contato 5	Chave traseira
P4.5	Disparo de sonar	Caso se conecte um sonar do tipo *Polaroid 6500*, este pio pode ser usado para disparar o sonar
P5.0	Foto-resistor 1	Posição sugerida: dianteira esquerda
P5.1	Foto-resistor 2	Posição sugerida: dianteira central
P5.2	Foto-resistor 3	Posição sugerida: dianteira esquerda
P5.3	Foto-resistor 4	Posição sugerida: em cima do braço robótico
P5.5	Foto-resistor 5	Posição sugerida: traseira
P5.6	Microfone	
P5.7	Medidor de nível da bateria	Indica quando a bateira abaixa de 6,5 Volts

12.1.3 Operação

Um ciclo normal de operação do robô tem os seguintes estágios:

- Desenvolvimento do comportamento no PC
- Conexão do robô ao PC via cabo serial
- Download do comportamento para o robô
- Desconexão do robô do PC
- Operação normal do robô

12.1.3.1 Desenvolvimento do comportamento no PC

Este procedimento é melhor detalhado na parte que versa sobre *software* neste manual. Após o comportamento estar desenvolvido, o usuário deve ter uma arquivo com a extensão *HEX*, que deve ser enviado ao robô.

12.1.3.2 Conexão do robô ao PC via cabo serial

Conectar uma das pontas do cabo serial no PC. Conectar a outra ponta (necessariamente DB9 fêmea) no robô. Seguir a lista de procedimentos:

- Certificar-se que a chave de seleção da placa está na posição “PC”

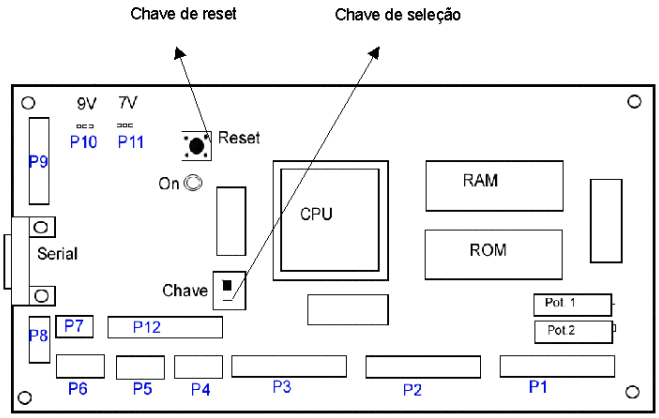


***Figura 38 - Envio de programa ao robô.* Créditos: Fr&d**

- Rodar o programa WinTalk, no PC

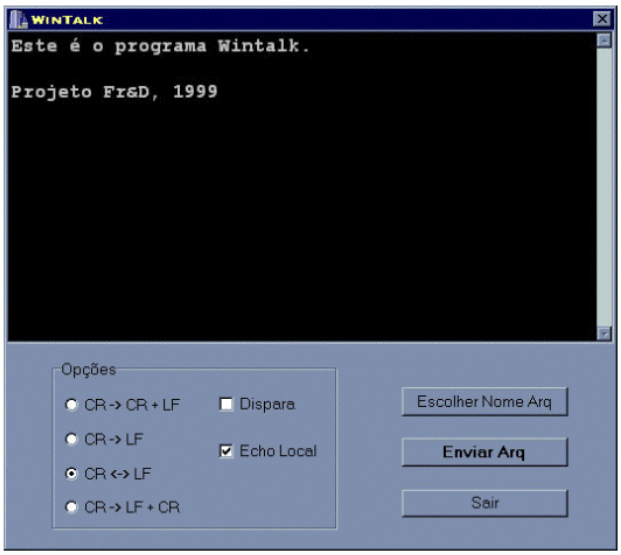


***Figura 39 - O programa WinTalk para envio de programas pela serial.* Créditos: Fr&d**

- Ligar o robô, na chave que fica no local indicado na figura abaixo
- Apertar a chave de *reset* do robô, que fica no local indicado na figura abaixo
- Deve aparecer na tela a mensagem “PCS 599 – Laboratório de Microprocessadores”. Esta mensagem vem do programa monitor modificado que é utilizado para envio do programa ao robô. Esta mensagem vem de um prOgrama que roda no robô, e não é gerada no PC
- O usuário deve clicar no botão “Escolher Nome Arq”, e selecionar o arquivo *HEX* gerado durante a edição do comportamento
- Clicar no console (onde aparecem as mensagens) e digitar “L” (atenção: precisa ser um “L” maiúsculo). Aparecerá na tela a mensagem “Leitura HEX”
- Deve-se então escolher a opção “Enviar Arq”, e o comportamento será enviado ao robô

12.1.3.3 Desconexão do robô do PC

Antes de desconectar, o usuário deve mudar a posição da chave mencionada no item acima. Para que o robo execute o comportamento, ela deve ser tirada da posição “PC”.

Depois de seguir os passos do item acima, é preciso desconectar o cabo serial (da ponta que fica no robô) e aguardar 10 segundos para que o robô comece a executar.

Se precisar fazer o *reset* do circuito por algum motivo, o usuário deve clicar na chave de reset e refazer o download do comportamento para o robô.

13. Manual de Manutenção

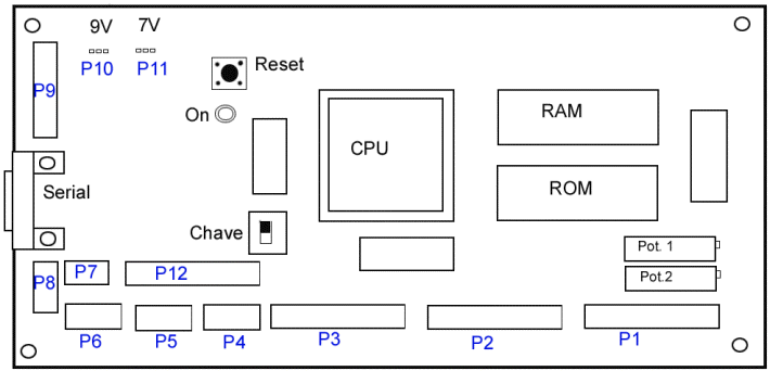


Figura 40 - Esquema da placa principal de controle. **Créditos: Fr&d**

O que fazer se...

...as fitas adesivas se soltarem?

Toda a estrutura do braço é afixada com fitas adesivas de dupla face. No caso de a fita se soltar ela deve ser substituída. Ela pode ser encontrada em papelarias grandes ou em casas especializadas em material de modelismo. São vendidas em rolos pequenos (alguns metros). A fita, além de possuir as duas faces adesivas, deve possuir uma fina camada de espuma entre elas. Esta camada serve para dar uma certa flexibilidade de movimentos e amortecer impactos. Caso a fita adquirida tenha uma das faces menos colante que a outra, pode-se usar duas camadas de modo que as duas faces menos colantes fiquem grudadas entre si e as duas mais colantes fiquem para fora.

...os parafusos da garrinha se soltarem?

Estes parafusos não devem ser muito apertados pois podem travar o movimento da garra. A porca deve ser atarrachada de modo a não exercer pressão nas articulações. Para que ela não se solte é interessante pingar uma gota de cola branca fixando a porca no parafuso. Deve-se tomar o cuidado de não deixar a cola escorrer para a articulação. Não é recomendável utilizar-se Super-Bonder (ou outras colas definitivas), pois além de ela ser muito líquida pode impedir uma futura desmontagem.

...braço se movimenta muito lentamente ou só treme?

É muito provável que a pilha esteja gasta. É recomendável utilizar pilhas alcalinas na substituição, pois o conjunto de motores drena uma corrente alta demais para pilhas comuns ou recarregáveis.

...circuito reseta (reinicia) sozinho?

Provavelmente as pilhas estão fracas e a corrente exigida pelos motores faz a alimentação do circuito não ser suficiente. É recomendável utilizar pilhas alcalinas na substituição.

...parte mecânica está inoperante?

Deve-se, em primeiro lugar, verificar se o cabo que interliga a parte mecânica ao circuito principal (P9) não esta rompido ou ligado invertido. Caso isto não tenha ocorrido, o problema deve estar na alimentação do *Mini SSC II* (bateria de 9V). O cabo (P10) pode estar solto, invertido ou rompido, ou ainda a bateria pode estar descarregada. É recomendável utilizar-se baterias alcalinas. É importante tomar o cuidado de não ligar a bateria invertida nem mesmo por poucos instantes pois isto pode queimar o Mini SSC II.

...programa não estiver sendo transferido para o robô?

Em primeiro lugar deve-se verificar se o a chave de seleção da placa (Chave) está na posição PC. Em seguida o cabo serial que liga o PC no robô (Serial) deve ser verificado. Os conectores devem estar bem encaixados. O problema pode ainda ser na parte de software. Fecha-se o programa de comunicação com o robô, iniciando-o novamente.

...algum sensor não está funcionando?

O cabo do sensor pode estar solto, com mau contato, rompido ou ainda invertido.

...o LDR (sensor de luminosidade) funciona mas não está respondendo adequadamente?

Provavelmente os potenciômetros de regulagem do conversor A/D não estão corretamente calibrados. Com uma peque chave de fenda deve-se ir girando-os até obter a regulagem adequada. O potenciômetro *Pot 1* é determina o limite inferior da conversão e o *Pot 2* determina o limite superior. Deve-se alterar as regulagens e ao mesmo tempo submeter a sensor a testes no escuro e claro extremos.

14. Considerações Finais

14.1 Avaliação

Finalmente a Escola Politécnica nos oferece, em nosso último ano, a chance de conduzir um projeto inteiro, desde os esboços iniciais até a sua entrega em funcionamento, passando por especificação, projeto e implementação. A forma como foi proposto aos alunos permite empregar boa parte dos conceitos teóricos aprendidos no curso de Engenharia, além de proporcionar uma bem acolhida vivência prática.

Passamos por todas as fases de um projeto de Engenharia, adquirindo conhecimento e experiência preciosos sobre como conduzir um projeto. Desde a euforia inicial até a afobação quando se aproxima o prazo de entrega, aprendemos lições valiosas de organização, metodologia de trabalho, cooperação, trabalho em equipe, criatividade e força de vontade.

Sobre o nosso projeto, tivemos que ter bastante versatilidade para analisar alternativas novas que apareciam e contornar dificuldades ou erros de avaliação quanto a complexidade de propostas feitas na especificação. À medida que fazíamos o desenvolvimento, nosso conhecimento de domínio aumentava, e podíamos ver falhas ou incoerências no projeto, aperfeiçoando-o. Nossa equipe se manteve bastante unida durante todo o processo, o que atenuou os momentos de tensão e nos educou no trabalho em equipe.

Pretendemos divulgar o máximo de informações produzidas e/ou coletadas por nós durante este projeto na *Internet*, para que futuros criadores de robôs possam encontrar menos pedras pelo seu caminho. Gostaríamos, também, que nosso projeto fosse continuado e melhorado nos próximos anos por outras equipes de projeto de formatura que se interessem por robótica móvel e modelos de comportamento. Um bom caminho para começar seria a lista de sugestões que damos no ítem abaixo

14.2 Alternativas

- Ao invés de servomecanismos, o custo poderia ser barateado se usássemos um motor DC e fizéssemos o controle utilizando o contador de pulsos do *80C552* e um *shaft encoder*. O motor seria controlado pelas portas PWM do microcontrolador, como vemos em [Jones&Flynn93]. O esquema atual emprega um *loop localmente realimentado*, o que não nos dá um controle de alto nível sobre os resultados das atuações.

- Um microcontrolador mais adequado seria o *80C562*, do ponto de vista de redução de custos. Suas diferenças em relação ao que foi usado são conversores A/D de 8 bits (ao invés de 10 bits) e ausência de suporte ao protocolo I^2C. Ele é 30% mais barato que o *80C552*.

- Um sistema de multitarefas *preemptivo* pode garantir que o robô continue a operar mesmo se um dos processos eventualmente entrar em laço infinito.

- O uso de bateria recarregáveis proporcionaria uma maior economia para os usuários ao longo do tempo. Para viabilizar seu uso, seria necessário reduzir bastante o consumo médio de corrente do robô. Se o recarregador pudesse ser embarcado, o robô poderia operar sem ser desligado por grandes períodos de tempo.

- A chave de seleção de canal serial entre PC e Mini SSC poderia comutar automaticamente, sem necessidade de intervenção do usuário para o início da execução.

- Poderia ser usado um microcontrolador com razoável quantidade de memória interna (*RAM* e *xROM*) para diminuir a área da placa principal. Isto facilitaria a confecção de robôs menores e aumentaria o número de portas disponíveis.

- Um ambiente gráfico mais fácil de usar poderia ser empregado, permitindo que usuários mais leigos criem comportamentos, como em [*Lego*].

- Um sistema auxiliar de alimentação com pequenas baterias (com as de relógio) poderia permitir que a *CPU* e a memória *RAM* não fossem reiniciados sempre que o circuito fosse desligado.

- A *API* de acesso aos serviços do robô pode ser implementada em outras plataformas, assim o sistema poderia ser portado, por exemplo, para robôs caminhantes (que usam patas).

- O uso de rádio adicionaria uma série de funcionalidades ao projeto, como comunicação entre robôs, permitindo o estudo de comportamentos coletivos. Poderia permitir ainda a intervenção do usuário, controlando parcialmente o robô.

- A instalação de micro câmeras e emissores de baixa potência poderia permitir aplicações de telepresença.

- A placa de controle do robô poderia ser substituída em parte por um *handheld device* (*palmtops*). A interface com os sensores e atuadores continuaria sendo feita por circuito dedicado, porém a programação e interface com o usuário seriam feitas pelo aparelho. Geralmente, tais aparelhos possuem um visor LCD sensível a toque e um canal serial para comunicação com o computador e a interface dos sensores e atuadores.

15. Referências

15.1 Bibliografia Comentada

[BorensteinA] BORENSTEIN, J.; EVERETT, H.R.; FENG, L.; WEHE, D. **Mobile Robot Positioning - Sensors and Techniques.** Journal of Robotic Systems, Special Issue on Mobile Robots, Vol 14 No 4, p.231-249.

[BorensteinB] BORENSTEIN, Johann; KOREN, Yoram. **Motion Control Analysis of a Mobile Robot.** Journal of Dynamics, Measurement and Control, Vol 109, No 2, p.73-79.

Os autores de ambos os artigos são reconhecidos pelas pesquisas na área de posicionamento e navegação robótica. Utilizamos para criar parte dos algoritmos de navegação do robô.

[Brooks86] BROOKS, Rodney A. **A Robust Layered Control System for a Mobile Robot.** IEEE Journal of Robotics and Automation, RA-2, p.14-23, April (1986) 14-23.

Os primeiros fundamentos do modelo de subsumption (ainda sem tal denominação) são apresentados. Rodney Brooks baseava sua estrutura de programação em camadas independentes e com finalidade distintas. Descreve uma implementação usando uma rede microprocessadores de 8 bits.

[Brooks89] BROOKS, Rodney A. **A Robot that Walks: Emergent Behavior Form a Carefully Evolved Network.** Neural Computation, 1(2) (Summer 1989) 253-262.

Melhor estruturação do modelo de subsumption, apresentando praticamente todas as características que hoje compõem o modelo. Vital para o entendimento do conceito de subsumption e interessante modelamento de comportamento animal pelo modelo.

[Brooks90a] BROOKS, Rodney. **The Behavior Language; User's Guide**. A.I. Memo 1227. MIT Press, Cambridge, MA, 1990.

Normatização de uma linguagem de programação baseada em LISP para modelamento de comportamentos utilizando subsumption. Definição de termos representando algumas das estruturas discutidas em outros artigos, como os hormones.

[Brooks90b] BROOKS, Rodney A. **Elephants Don't Play Chess**, in (**Designing Autonomous Agents: Theory and Pratice from Biology to Engineering and Back**. Ed Patie Maes, MIT Press, MA, 1990). p3-15, 1990.

Apresenta o modelo de subsumption como uma ramo da inteligência artificial que difere daquilo que já foi pesquisado nos últimos trinta anos. Mostra as linhas de pesquisa em andamento e os projetos futuros.

[Brooks91a] BROOKS, Rodney A.7 **Intelligence Without Reason.** AI Memo No 1293 (for Computer and Thoughts, IJCAI-91), MIT Press, Cambridge, MA, 1991.

Mostra como o modelamento de sistemas com subsumption se aproximam dos sistemas biológicos melhor do que os métodos convencionais.

[Brooks91b] BROOKS, Rodney A. **Intelligence Without Representation.** Artifical Intelligence 47 (1991), 139-159, MIT Press, Cambridge, MA, 1991.

Mostra como o uso de subsumption evita a necessidade de se modelar todo o ambiente em que o robô (ou agente) se encontra. Mostra como camadas de "instintos" conseguem ter uma reação aos estímulos exteriores com melhor desempenho do que os métodos atuais utilizados pela inteligência artificial.

[Brooks91c] BROOKS, Rodney A. **New Approaches to Robotics**. MIT Press, Cambridge, MA, 1991.

Comparativo entre os vários meios de construção de sistemas inteligentes, apresentando vários exemplos e analisando suas vantagens e desvantagens.

[Calvert97] CALVERT, C. **Borland C++ Builder 3 Unleashed.** Sams Publishing & Borland Press, 1998.

Uma enciclopédia de informações para aplicações sérias utilizando o C++Builder 3.0. Enfoque principal em banco de dados, cobrindo também componentes para Internet, computação gráfica e arquiteturas de comunicação entre objetos.

[Connell89] CONNELL, Jonathan H. **A Colony Architecture for an Artificial Creature.** MIT Ph.D. Thesis in Electrical Engineering and Computer Science, MIT AI Lab Tech Report 1151 (June 1989).

Apresenta uma interessante proposta de utilizar grupos de robôs baseados em subsumption para explorarem e colonizarem ambientes.

[Connell90] CONNELL, Jonathan H. **Minimalist mobile robotics: a colony-style architecture for an artificial creature**. Academic Press Inc, 1990.

Mais discussão sobre o uso de pequenos robôs e em grande número para colonização.

[Doty96a] DOTY, Keith L.; IGLESIA, Erik de la. **Sharp IR Sensor Hack for Analog Distance Measurement.** EEL 5934, Dept Electrical Engineering, University of Florida, EUA, 1996.

Alteração nos sensores Sharp GPIU58X ou GPIU58Y para desabitar o Schmitt trigger e possibilitar a deteção de distâncias através do nível de tensão na saída do receptor infravermelho.

[Doty96b] DOTY, Keith L.; JANTZ, Scott. **Talrik II Users Manual.** Mekatronix, Gainesville, FL, 1996.

Manual completo de operação do robô Talrik II da Mekatronix, apresentando todos os circuitos e modos de operação.

[Doty97a] DOTY, Keith L. **Servo Hack: Mekatronix MS410 and MS455.** Mekatronix paper, Gainesville, Florida, EUA. 1997

Apesar de ser específico para um modelo de servo, possui uma ilustração muito interessante de um servo desmontado e com seus componentes. Facilita a identificação do potenciômetro durante a alteração.

[Doty97b] DOTY, Keith L. **Talrkik II Assembly Manual.** Mekatronix,Gainesville, FL, 1997.

Manual completo de montagem do robô Talrik II da Mekatronix, inclusive com dicas de soldagem e corte de materiais para a estrutura mecânica.

[Iovine98] IOVINE, John. **Robots, Androids and Animatrons: 12 Incredible Projects You Can Build**. McGraw-Hill, 1998.

Interessante livro com vários projetos prontos e detalhados para serem implementados. Vários circuitos foram pesquisados e alguns incluídos em nosso robô.

[Jones&Flynn93] JONES, Joseph L.; FLYNN, Anita M. **Mobile Robots: Inspiration to Implementation**. Editora A K Peters, 1993.

Simplesmente o melhor livro que conseguimos sobre robótica. Apresenta todos os conceitos importantes sobre robôs móveis, com extensa discussão sobre as diferentes tecnologias e dois projetos completos (com circuito e software) de robôs móveis. É o ponto inicial, e muitas vezes suficiente, para entusiastas de robótica de todos os níveis.

[Lynxmotion98a] LYNXMOTION. **5 Axis Robot Arm Kit Rev. 2.** Assembly Manual 5AA-0-7 Ver 6.0. Lynxmotion Inc, Pekin, IL, EUA, 1998.

Manual de montagem do braço mecânico de 5 eixos. Possui todos os passos explicados e bem ilustrados. Importante seguir os passos na ordem indicada, assim como observar a orientação das peças através das pinagens.

[Lynxmotion98b] LYNXMOTION. **Mobile Arm Robot Kit.** Assembly Manual MRA-0-3 Ver 3.0. Lynxmotion Inc, Pekin, IL, EUA, 1998.

Manual de montagem da plataforma móvel do robô e a sua integração com o braço mecânico.

[Lynxmotion98c] LYNXMOTION. **Servo Information and Modification.** Users Manual SERVO-0-3 Ver 2.0. Lynxmotion Inc, Pekin, IL, EUA, 1998.

Manual de operação e modificação dos servos Hitec incluídos no kit. Além de esquemas úteis para a conexão ao SSC, possui as modificações necessárias para a transformação em servomotor.

[Lynxmotion98d] LYNXMOTION. **Tracker – Line Tracking Kit.** Assembly Manual TMRA-0-3 Ver 2.0. Lynxmotion Inc, Pekin, IL, EUA, 1998.

Manual de montagem do kit rastreador. Possui circuito interno, exemplos de programas e modo de montagem e acoplamento ao robô.

[Matai98] MATAI, Shigueharu. **Recomendações para a Elaboração do Relatório de Estágio.** Apostila da disciplina PCS-600 Estágio Supervisionado, PCS/USP, 1998.

Foi nosso primeiro contato com uma maneira mais acadêmica de redigir relatórios. Apesar de possuir apenas nove páginas, contém dicas interessantes. A complementação desse modo de redação foi feita pelos professores do Laboratório de Projeto de Formatura.

[Martin91] MARTIN, Fred. **The MIT Sensor Robot: User's Guide and Technical Reference.** MIT Press, Cambridge, MA, 1991.

Manual sobre uma implementação de robô móvel feita no MIT utilizando diversos sensores.

[Martin93] MARTIN, Fred. **Electronic Brick Technical Notes.** Learning and Epistemology Group, The Media Laboratory, MIT, Cambridge, MA, 1993.

Descrição da estrutura interna dos blocos funcionais utilizados pelo Lego MindStorms.

[McComb87] McCOMB, Gordon. **The Robot Builder's Bonanza: 99 Inexpensive Robotics Projects**. TAB Books, a McGraw-Hill division, 1987.

Apesar de um pouco antigo, possui diversos projetos (principalmente mecânicos) úteis para qualquer projeto de robótica.

[Miano97] MIANO, J.; CABANSKI, T; HOWE, H. **Borland C++ Builder How-To.** Waite Group Press, 1997.

Interessante lista de truques interessantes e indispensáveis para qualquer programa em C++Builder. Apesar de ter sido escrito para a versão 1.0, ainda é útil até mesmo para versão 4.0.

[Philips98] PHILIPS SEMICONDUCTORS. **87C552 Data Sheet.** Order number 9397 750 05367, Philips Eletronics North America, USA, 1998.

Dentre os diversos datasheets que utilizamos, este é fundamental para o entendimento do microcontrolador utilizado.

[Philips94] PHILIPS SEMICONDUCTORS. **Using the analog-to-digital converter of the 8XC552 microcontroller.** Application Note EIE/AN93017, Philips Electronics North America, USA, 1994.

Um dos poucos complementos necessários ao datasheet do microcontrolador. Indica com exemplos o algoritmo para se realizar as conversões. Atenção para os NOPs necessários e delays.

[Raymond98a] RAYMOND, E. **The Cathedral and the Bazaar.** August 1998.

Uma revolução no modo de se programar, aproveitando-se os recursos que a Internet oferece.

[Raymond98b] RAYMOND, E. **Homesteading the Noosphere**. April 1998.

O estranho comportamento de hackers que utilizam o método Bazaar de programação.

[Reisdorph98] REISDORPH, K. **Borland C++ Builder 3 in 21 Day.** Sams Publishing & Borland Press, 1998.

Excelente livro para quem deseja iniciar seus estudos em C++Builder ou tirar aquelas pequenas dúvidas sobre algum tópico.

[Scott] SCOTT EDWARDS ELECTRONICS. **Using the Mini SSC II (Serial Servo Controller).** Scott Edwards Electronics Inc, Sierra Vista, AZ, EUA.

Manual do usuário do Mini SSC II, contendo instruções para conexão com microcontroladores e PCs, além de exemplos de programas.

[TSS97] TECHNICAL SERVICE & SOLUTIONS. **Build and Using the IRPD.** Technical Service & Solutions. Lynxmotion Inc, Pekin, IL, EUA, 1997.

Manual do kit IRPD doado pela Anacom. Possui instruções de montagem, circuito do kit e exemplos de programas para acessá-lo.

[Wickelgren96] WICKELGREN, Ingrid. **Ramblin' Robots: Building a Breed of Mechanical Beasts**. Ed Watts, 1996.

Interessante compilação de robôs alternativos utilizando as mais estranhas técnicas. Um excelente índice de idéias para quem deseja inovar e se aventurar em projetos ousados.

15.2 Sites na Internet

[*AUVS*] ***Association for Unmanned Vehicle Systems (AUVS) International Robotics Competition***.

http://avdil.gtri.gatech.edu/AUVS/IARCLaunchPoint.html

Nosso sonho ainda é participar desta competição.

[*DPRG*] ***DPRG - Dallas Personal Robotics Group***. http://www.dprg.org

Um dos vários grupos de robótica amadora que nos inspiraram a criar o nosso próprio.

[*HVW*] ***HVW Technologies: All About Servos.*** http://www.hvwtech.com/about_servos.htm

Pequeno tutorial sobre servos e modos de transformar servomecanismos em servomotores. Para modelos Futaba FP-S148 e TS-53.

[*Johuco*] ***Johuco Ltd***. http://www.pcrealm.net/~johuco/

Kits de robótica simples para hobistas de vários níveis.

[*Lego*] ***Lego MindStorms***. http://www.legomindstorms.com

O mais flexível e popular kit de robótica hoje em dia. Muito bem acabado, utilizando peças de alta qualidade e o apelo tradicional dos blocos de vários tipos que a Lego possui.

[*Lynxmotion*] ***Lynxmotion Robot Kits***. http://www.lynxmotion.com

O kit que utilizamos em nosso robô. Excelente plataforma mecânica para ser integrada em qualquer microcontrolador e tipos de sensores. Possui alguns opcionais interessantes para a criação robôs muito mais ágeis e com controle mais preciso.

[*Mekatronix*] ***Mekatronix***. http://www.mekatronix.com

Outro fabricante interessante de kits de robótica utilizando uma plataforma de madeira. Possui vários sensores e parece ser customizável.

[*Newton*] ***Newton Research Labs***. http://www.newstonlabs.com

Robôs para aplicações científicas, de alta performance e de alto custo. Possui o estado de arte em sensores e construção mecânica.

[*Novasoft*] ***NovaSoft***. http://www.mrrobot.com

Kits para iniciantes em robótica.

[*Open*] ***OpenSource Foundation***. http://www.opensource.org

A filosofia de distribuição de software que nos baseamos nosso futuro site sobre robótica. Outra revolução em software que está cada vez mais evidente ser necessária no mundo capitalista.

[*Pioneer*] ***Pioneer Mobile Robots***. http://www.activmedia.com/robots/

Mais robôs profissinais, com diversos opcionais de ótima desempenho.

[*SKC*] ***Science Kit Center***. http://www.sciencekits.com/robots.htm

Kits para crianças que começam a demonstrar interesse por robótica.

[*Wars*] ***Robot Wars Official Site***. http://www.robotwars.com

Robôs controlados por humanos que tentam destruir um ao outro. Interessante.

Entre dezenas de outros sites que pode-se achar na Internet buscando por “`robotics`” ou “`mobile robots`”. Em especial, destacamos os sites de laboratórios de pesquisas acadêmicos e governamentais, como o *AI Labs* do *MIT* e diversos dentro da própria *NASA*.

16. Resumo do Projeto

ROBÔS MÓVEIS PARA O MODELO DE SUBSUMPTION

Professor: Marco Túlio Carvalho de Andrade
mtulio@pcs.usp.br
Equipe: Daniel Carlos Landi
daniel.landi@poli.usp.br
Fábio Roberto de Miranda
fmiranda@lsi.usp.br
Rodrigo Stephan de Souza Telles
r-telles@usa.net

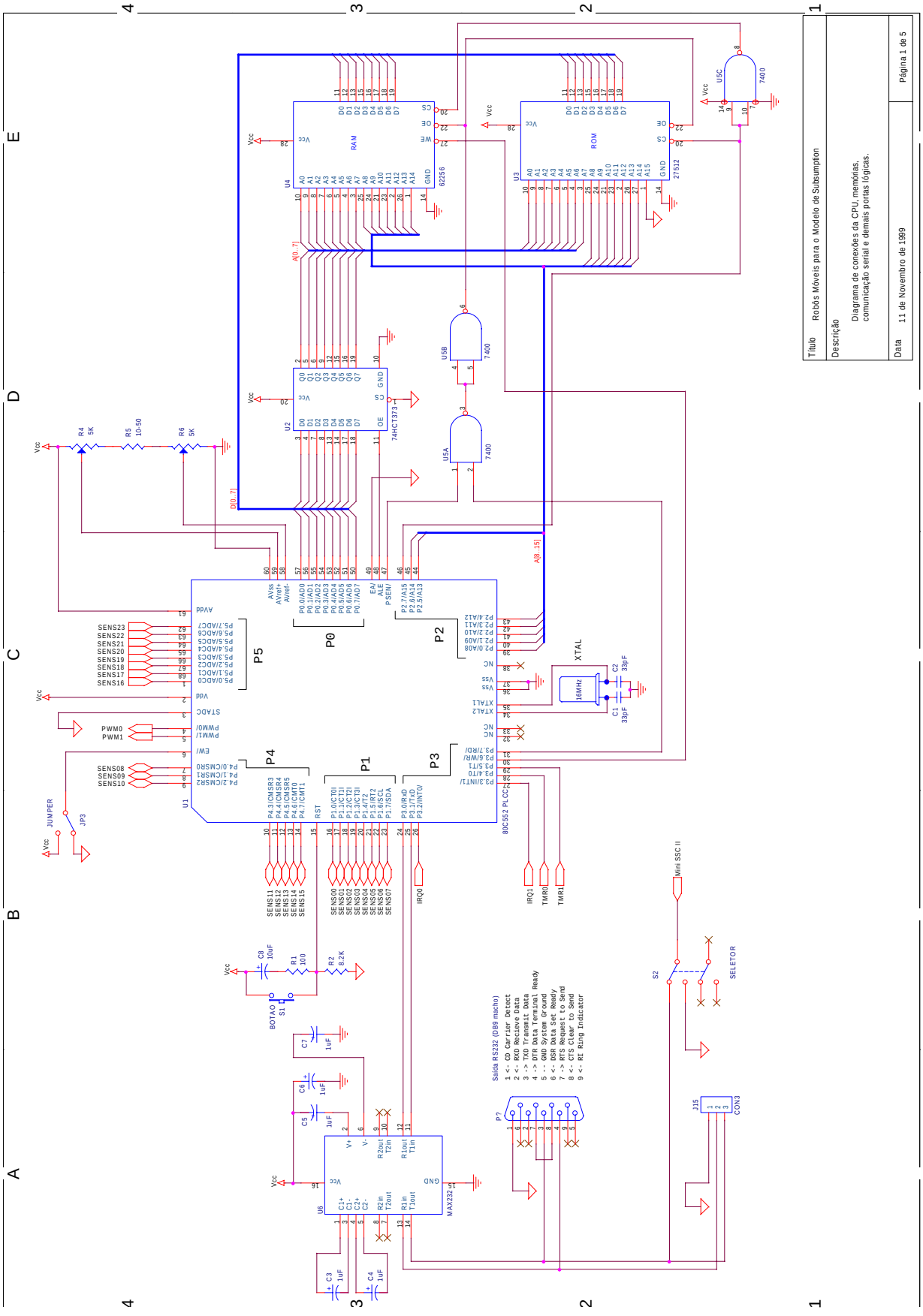
RESUMO

Visamos com nosso trabalho de formatura projetar e implementar uma arquitetura de robôs móveis que possa ser utilizada para fins didáticos ou como introdução à robótica para interessados na área. Visando a popularização, manteremos o projeto flexível e bastante diversificado, permitindo que diferentes componentes possam ser testados e utilizados. Para diferenciar nosso projeto dos demais *kits* comerciais, concentraremos grande esforço na elaboração de ferramentas para programação de comportamentos do robô. Propomos o projeto de um robô movido por duas rodas diferenciais e dotado de um microcontrolador dedicado da família 80C51. Não há qualquer comunicação com o computador, logo todo o algoritmo de controle e de decisão estará armazenado na memória interna. Uma ilustração simples do robô pode ser vista na página em anexo. O 80C51 escolhido possui saídas PWM e conversores A/D, além de outras facilidades que tornarão o circuito compacto. Os sensores previstos são de vários tipos: fotocélula, detetor digital e analógico de infravermelho, microfone, sonar ultrasônico, sensor piroelétrico e sensor de contato. Os atuadores se restringem a motor DC contínuo, LED e alto-falante. Vários desses componentes são intercambiáveis, permitindo que o robô seja configurado para diversas aplicações. A programação do robô deverá focar diversos tipos de usuários. Implementamos uma linguagem baseada em *subsumption* para a programação dos comportamentos. O item mais importante na programação será o uso do modelo de *subsumption*. Trata-se do método de programação de robôs voltados para pesquisa que mais ganha força, revolucionando a área há alguns anos atrás. O comportamento do robô é controlado por uma rede de processos que são executados em paralelo. Cada processo possui sua finalidade e tem acesso às entradas (sensores) e às saídas (atuadores) do robô. Eles também possuem diferentes prioridades. Um processo pode suprimir o resultado de outros processos menos prioritários quando necessário. As principais características são: modularidade, escalabilidade e robustez.

17. Anexos

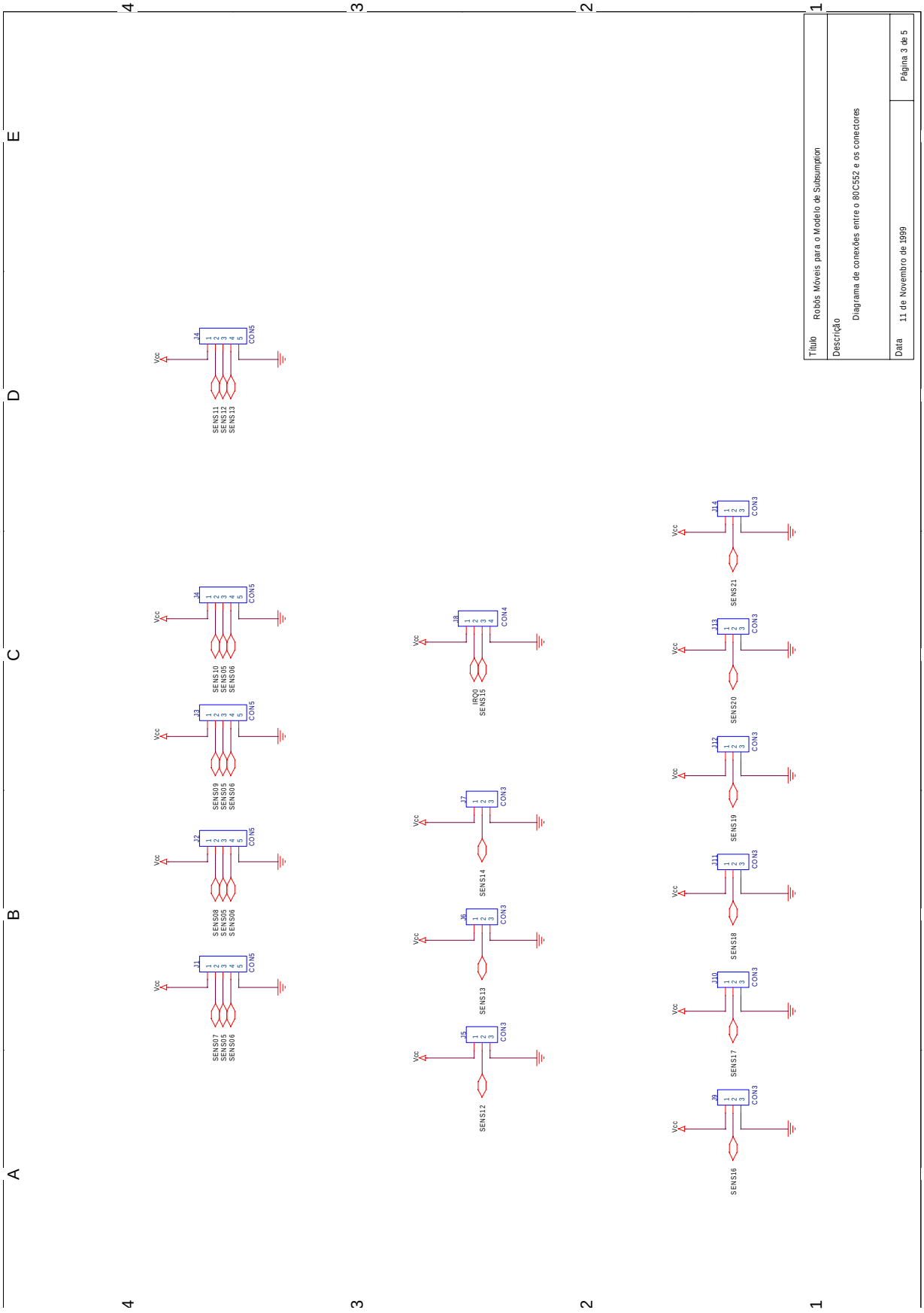
17.1 Anexo I

Esquema Elétrico: Microcontrolador, Memórias e Canal Serial



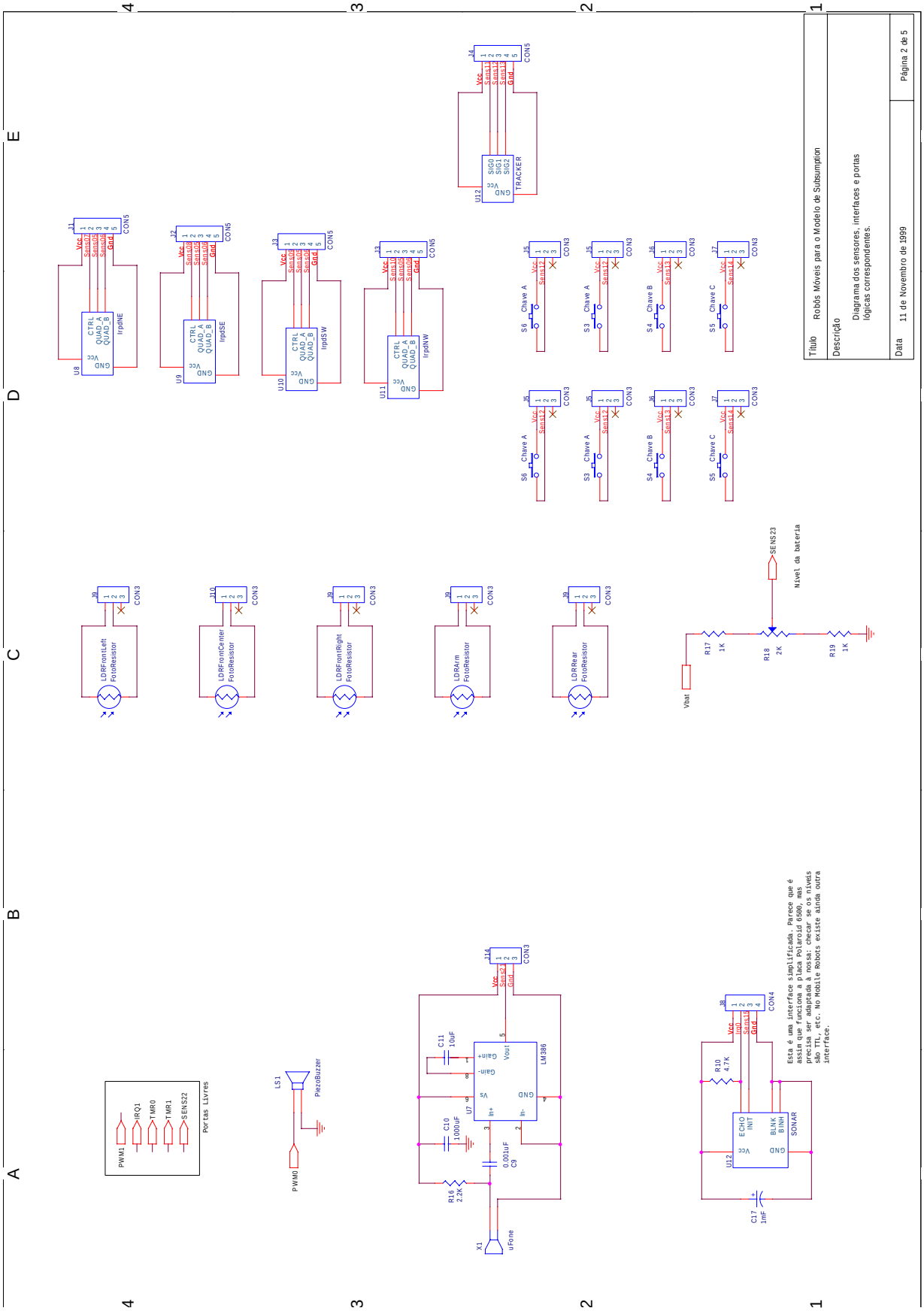
17.2 Anexo II

Esquema Elétrico: Conectores



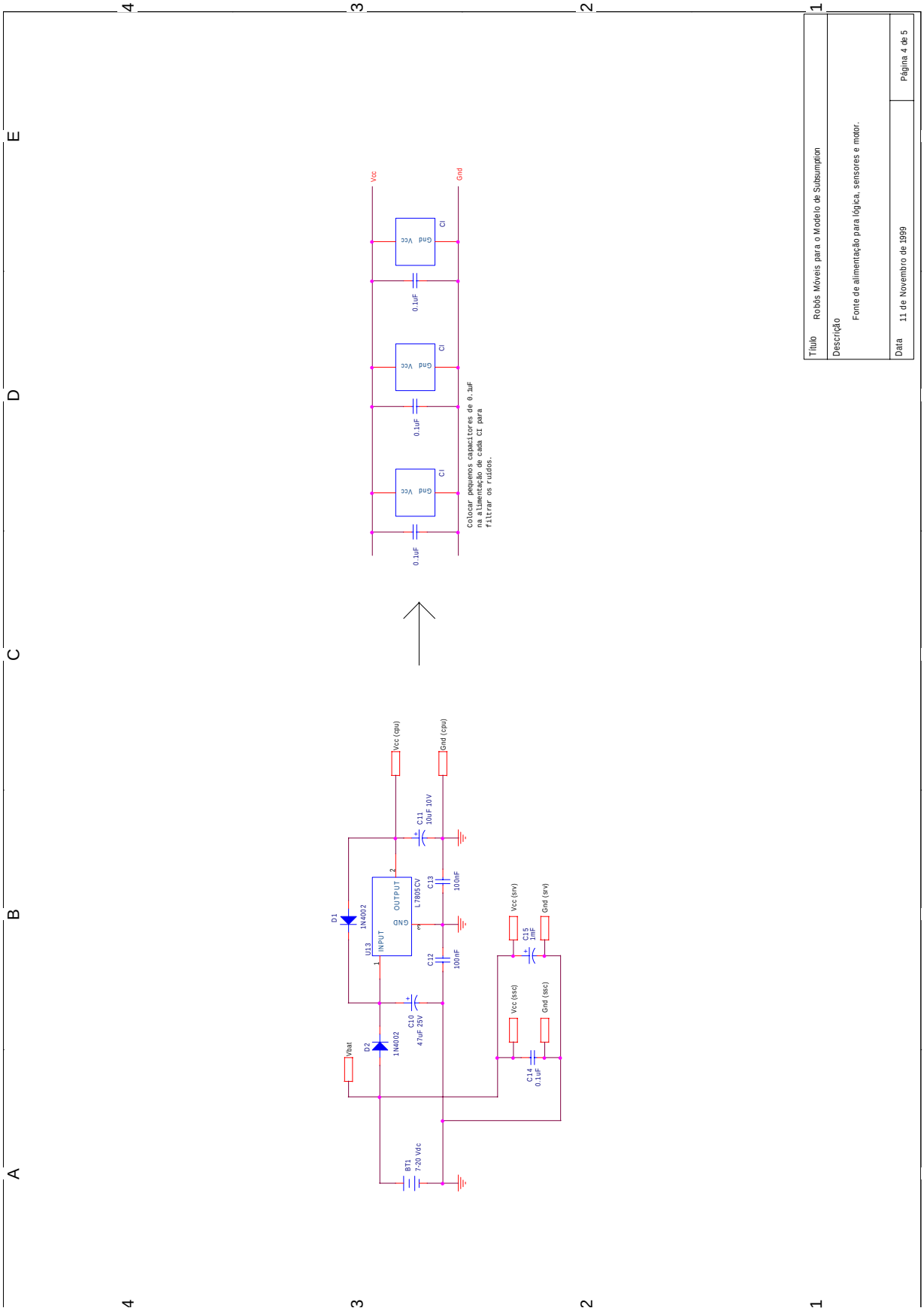
17.3 Anexo III

Esquema Elétrico: Sensores



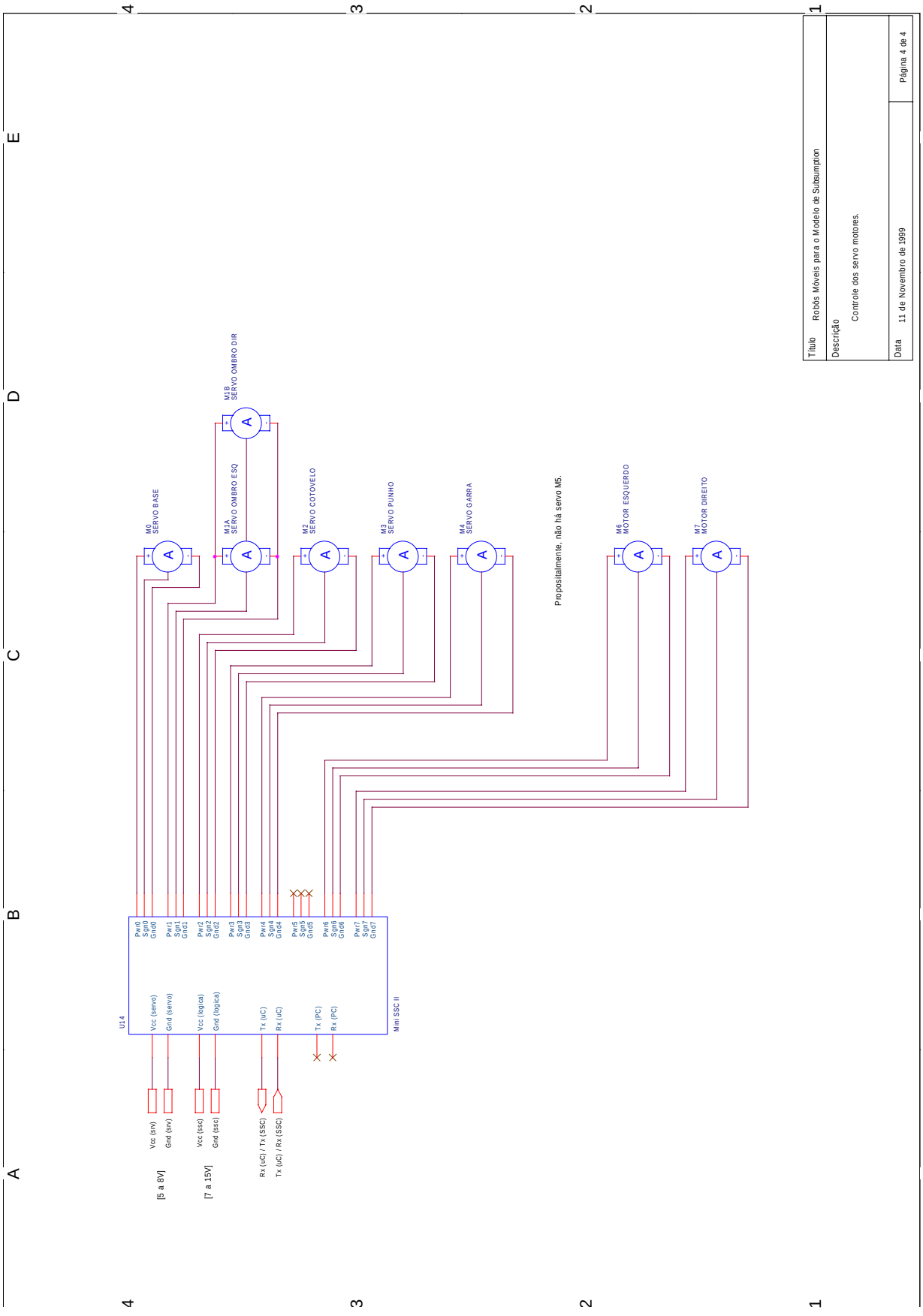
17.4 Anexo IV

Esquema Elétrico: Circuitos de Potência



17.5 Anexo V

Esquema Elétrico: Servos e Motores



17.6 Anexo VI

Exemplo de programa em linguagem *SOUL*, conforme apontado no item *6.1.1*:

```
process Andar step 5

    outputs: MotorDir, MotorEsq;

    MotorDir = 177;
    MotorEsq = 177;

    return;

process EvitarObstaculo step 2

    inputs: IRPD0, IRPD1, IRPD2, IRPD3;
    outputs: MotorDir, MotorEsq;
    locals: Counter, Direcao;

    stateInicial:
        if (IRPD0 || IRPD1) {
            Direcao = 0;
            next = stateVoltar;
        }


        if (IRPD2 || IRPD3) {
            Direcao = 1;
            next = stateVoltar;
        }


    stateVoltar:
        MotorDir = 27;
        MotorEsq = 27;

        Counter = 10;

        next = stateEsperaVoltar;

    stateEsperaVoltar:
        MotorDir = 27;
        MotorEsq = 27;

        if (—Counter == 0) {
            next = stateVirar;
        }

    stateVirar:
        if (Direcao) {
            MotorDir = 77;
            MotorEsq = 177;
        } else {
            MotorDir = 177;
            MotorEsq = 77;
        }

        Counter = 10;

        next = stateEsperaVirar;
```

```
    stateEsperaVirar:
        if (Direcao) {
            MotorDir = 77;
            MotorEsq = 177;
        } else {
            MotorDir = 177;
            MotorEsq = 77;
        }

        if (—Counter == 0) {
                next = stateInicial;
        }

process SegueLuz step 1

    inputs: LDR0, LDR1;
    outputs: MotorDir, MotorEsq;
    locals: Desvio, Trigger, Vezes;

    before
        Desvio = (LDR0 - LDR1);
        Trigger = (LDR0 > 30 || LDR1 > 30) &&
                  (Desvio > 20 || Desvio < -20);

    stateInicial:
        if (Trigger) {
            Vezes = Desvio / 5;
            next = stateCorrigeDir;
        }

    stateCorrigeDir:
        if (Desvio < 0) {
            MotorDir = 150;
            MotorEsq = 127;
        } else {
            MotorDir = 127;
            MotorEsq = 150;
        }

        if (Vezes— == 0) {
            next = stateInicial;
        }
}

connect:
    Andar.MotorDir to Fred.MotorRight priority 0;
    Andar.MotorEsq to Fred.MotorLeft priority 0;

    Fred.photoCell01 to SegueLuz.LDR0;
    Fred.photoCell02 to SegueLuz.LDR1;
    SegueLuz.MotorDir to Fred.MotorRight priority 1;
    SegueLuz.MotorEsq to Fred.MotorLeft priority 1;

    Fred.IR1Right to EvitarObstaculo.IRPD0;
    Fred.IR1Left to EvitarObstaculo.IRPD1;
    Fred.IR2Right to EvitarObstaculo.IRPD2;
    Fred.IR2Left to EvitarObstaculo.IRPD3;
    EvitarObstaculo.MotorDir to Fred.MotorRight priority 2;
    EvitarObstaculo.MotorEsq to Fred.MotorLeft priority 2;
```

No código anterior vemos a estrutura de descrição de um comportamento e de uma conexão entre processos. A palavra-chave `process` define o nome do processo. Após, temos a declaração das entradas (`inputs`), saídas (`outputs`) e variáveis locais (`Locals`). Duas palavras-chaves definem códigos que são sempre executados, antes (`before`) e depois (`after`) dos estados. Entre eles, existem os estados, que são executados de acordo com uma variável de estado corrente, e que contém as ações a serem executadas (em C) e a lógica de próximo estado.

A conexão entre estados é feita pela palavra-chave `connect`, que define qual saída de qual processo deve ser ligada a qual entrada de qual processp. A hierarquia de prioridades é aqui definida pela ordem em que as conexões aparecem e pela palavra-chave `priority`. Essa prioridade substitui o uso das estruturas `supress` em [Brooks90a] [Jones&Flynn93].

Vários grupos de `process`'s podem ser repetidos até descreverem todos os processos do robô. Ao final, a seção de `connect` estabelece as relações entre os processos, configurando o comportamento do robô.

17.7 Anexo VII

Conforme indicado no item *6.1.2*, para editarmos os comportamentos em *SOUL*, foi adicionada a seguinte descrição de linguagem ao arquivo `Wordfile.txt` do *UltraEdit-32*:

```
/L6"SOUL” Line Comment = // Line Comment Alt = ## Block Comment On = /* Block Comment Off = */
Escape Char = \ String Chars = “‘ File Extensions = BEH
/Delimiters = ~!@%^&*()-+=|\/{}[]:;”’<> , .?
/Function String = “%process ^([a-zA-Z_]*^) “
/Indent Strings = “{“
/Unindent Strings = “}”
/C1"C Keywords”
auto
break
case char const continue
default do double
else enum extern
float for
goto
if int
long
register return
short signed sizeof static struct switch
typedef
union unsigned
void volatile
while
__asm __fastcall __self __segment __based __segname __fortran __cdecl __huge __far __saveregs
__export __pascal __near __loadds __interrupt __inline
#define #error #include #elif #if #line #else #ifdef #pragma #endif #ifndef #undef
/C4"Operators”
+
-
=
// /
%
&
>
<
^
!
|
/C5"SOUL Keywords”
after
before
connect
input inputs
locals
next
priority process
output outputs
step
to
** state
/C6"Fr&d API”
** Fred
```

Maiores informações sobre tal estrutura podem ser encontradas na *Ajuda* (*Help*) do editor.

17.8 Anexo VIII

Com as desculpas consideráveis, segue a longa listagem de um processo compilado:

```
// Driver.c (antigo MegaDrive.c)
//
// Rotinas para acesso ao microcontrolador
// oferecendo ao programa de comportamento
// uma interface de controle apenas ‘logica’.
//

#include <80C552.h>
#include <stdio.h>

#define DEFAULT_DELAY   9000

// constant values for moving. real value: (x - 127) > 0 forward
// many approaches...                               < 0 backward

// ...motor speed oriented
#define FOLLOW_RIGHT_HARD  110 // fast right motor
#define FOLLOW_RIGHT_SOFT  116 // slow right motor
#define FOLLOW_LEFT_HARD   110 // fast left motor
#define FOLLOW_LEFT_SOFT   116 // slow left motor
#define FOLLOW_TURN_LEFT  2000 // left turn duration
#define FOLLOW_TURN_RIGHT 2000 // right turn duration

// ...turn direction oriented
#define ESCAPE_TURNR_RIGHT 140 // right turn for right motor
#define ESCAPE_TURNR_LEFT  147 // right turn for left motor
#define ESCAPE_TURNL_RIGHT 147 // left turn for right motor
#define ESCAPE_TURNL_LEFT  140 // right turn for left motor
#define ESCAPE_TURN_LONG  4000 // long turn duration
#define ESCAPE_TURN_SHORT 2000 // short turn duration

// ...the orientation is said in the constantes already
#define ESCAPE_FORWARD_RIGHT 110 // right motor forward
#define ESCAPE_FORWARD_LEFT  116 // left motor forward
#define ESCAPE_FORWARD      4000 // forward duration

// ...and so on (note that the backward is actually a back turn)
#define AVOID_TURNR_LEFT  110 // turn right for left motor
#define AVOID_TURNR_RIGHT 116   // turn right for right motor
#define AVOID_TURNR      2000 // turn right duration
#define AVOID_TURNL_LEFT  116 // turn left for left motor
#define AVOID_TURNL_RIGHT 110 // turn left for right motor
#define AVOID_TURNL      2000 // turn left duration
#define AVOID_BACK_LEFT   140 // backward for left motor
#define AVOID_BACK_RIGHT  147 // backward for right motor
#define AVOID_BACK       4000 // backward duration

// ...hard spin
#define SPIN_RIGHT  147 // spin for right motor
#define SPIN_LEFT   107 // spin for left motor
#define SPIN_TIME 10000 // spin duration

// ...finally
#define STOP        127 // stop for both motor
#define STOP_TIME 10000 // stop duration

// TH1 register reload for 9600baud serial comm @ 11.0592MHz
#define TH1_RELOAD  0xfd

// delay between writing and reading IRPD
#define IR_DELAY 25
```

```
// aux defs on A/D conv regs
#define ADCS ADCON_BITS.B3
#define ADCI ADCON_BITS.B4
#define ADEX ADCON_BITS.B5

// SFR var pointed to ADCON reg
static SFR_BITS ADCON_BITS @ 0xC5;

// generates a 800Hz wave at the PWM output
#define PWM_FREQUENCY  54
// and at 50% duty cicle
#define PWM_DUTY_CYCLE 85

// buzz off...
#define BUZZER_TIME 5

// motors' addresses
#define LEFT_MOTOR     6
#define RIGHT_MOTOR    7
#define BASE_MOTOR     0
#define SHOULDER_MOTOR 1
#define ELBOW_MOTOR    2
#define WRIST_MOTOR    3
#define GRIP_MOTOR     4

// step value for motors
#define MOTOR_STEP 1

// timer counters for processes
unsigned char Follow_counter;
unsigned char Avoid_counter;
unsigned char Escape_counter;

// the 'i' before the vars below are from 'internal'

/* IRPD global variables */
unsigned char iIrpdNWLeft;
unsigned char iIrpdNWRight;
unsigned char iIrpdNELeft;
unsigned char iIrpdNERight;
unsigned char iIrpdSWLeft;
unsigned char iIrpdSWRight;
unsigned char iIrpdSELeft;
unsigned char iIrpdSERight;

/* line tracker variables */
unsigned char iLineTrackerLeft;
unsigned char iLineTrackerCenter;
unsigned char iLineTrackerRight;

/* LDR variables */
unsigned char iLDRFrontLeft;
unsigned char iLDRFrontCenter;
unsigned char iLDRFrontRight;
unsigned char iLDRRear;
unsigned char iLDRArm;

/* bumper switches variables */
unsigned char iBumperFrontLeft;
unsigned char iBumperFrontRight;
unsigned char iBumperLeft;
unsigned char iBumperRight;
unsigned char iBumperRear;

/* microphone average input level variable */
unsigned char iMicrophoneLeft;
unsigned char iMicrophoneRight;
```

```
/* battery level sensing */
unsigned char iBattery;

/* buzzer time to sound */
unsigned long iBuzzerTime;

/* servomechanisms control */

/*  wheel motors */
unsigned char iLeftMotor;
unsigned char iRightMotor;


/*  arm motors */
unsigned char iBaseServo;
unsigned char iShoulderServo;
unsigned char iElbowServo;
unsigned char iWristServo;
unsigned char iGripServo;

/*  arm motors auxiliary variables */
unsigned char iBaseServoTarget;
unsigned char iShoulderServoTarget;
unsigned char iElbowServoTarget;
unsigned char iWristServoTarget;
unsigned char iGripServoTarget;

/*  motors delta values */
bit iLeftMotorChanged;
bit iRightMotorChanged;
bit iBaseServoChanged;
bit iShoulderServoChanged;
bit iElbowServoChanged;
bit iWristServoChanged;
bit iGripServoChanged;


// PERIPHERAL DATA ACCESSOR FUNCTIONS

/* IRPD methods */

unsigned char getIrpdNWLeft() {
    return iIrpdNWLeft;
}

unsigned char getIrpdNWRight() {
    return iIrpdNWRight;
}

unsigned char getIrpdNELeft() {
    return iIrpdNELeft;
}

unsigned char getIrpdNERight() {
    return iIrpdNERight;
}

unsigned char getIrpdSWLeft() {
    return iIrpdSWLeft;
}

unsigned char getIrpdSWRight() {
    return iIrpdSWRight;
}
```

```
unsigned char getIrpdSELeft() {
    return iIrpdSELeft;
}

unsigned char getIrpdSERight() {
    return iIrpdSERight;
}


/* line tracker functions */

unsigned char getLineTrackerLeft() {
    return iLineTrackerLeft;
}

unsigned char getLineTrackerCenter() {
    return iLineTrackerCenter;
}

unsigned char getLineTrackerRight() {
    return iLineTrackerRight;
}

/* LDR functions */

unsigned char getLDRFrontLeft() {
    return iLDRFrontLeft;
}

unsigned char getLDRFrontCenter() {
    return iLDRFrontCenter;
}

unsigned char getLDRFrontRight() {
    return iLDRFrontRight;
}

unsigned char getLDRArm() {
    return iLDRArm;
}

unsigned char getLDRRear() {
    return iLDRRear;
}


/* bumper switches functions */

unsigned char getBumperFrontLeft() {
    return iBumperFrontLeft;
}

unsigned char getBumperFrontRight() {
    return iBumperFrontRight;
}

unsigned char getBumperLeft() {
    return iBumperLeft;
}

unsigned char getBumperRight() {
    return iBumperRight;
}
```

```
unsigned char getBumperRear() {
    return iBumperRear;
}

/* microphone average input level variable */

unsigned char  getMicrophoneLeft() {
    return iMicrophoneLeft;
}

unsigned char  getMicrophoneRight() {
    return iMicrophoneRight;
}


/* iBattery level sensing */
unsigned char getBattery() {
    return iBattery;
}


/* setting buzzer time */
void setBuzzer() {
    iBuzzerTime = BUZZER_TIME;
}

/* servomechanisms control */

/*  wheel motors */
void setLeftMotor(unsigned char value) {
    iLeftMotor = value;
    iLeftMotorChanged = 1;
}

void setRightMotor(unsigned char value) {
    iRightMotor = value;
    iRightMotorChanged = 1;
}

/*  arm motors */
void setBaseServo(unsigned char value) {
    iBaseServoTarget = value;
}

void setShoulderServo(unsigned char value) {
    iShoulderServoTarget = value;
}

void setElbowServo(unsigned char value) {
    iElbowServoTarget = value;
}

void setWristServo(unsigned char value) {
    iWristServoTarget = value;
}

void setGripServo(unsigned char value) {
    iGripServoTarget = value;
}
```

```
// INITIALIZATION METHODS

/*
 * funcao generica para delay
 */

void delay(unsigned long int i) {

    unsigned long int j = 0;

    j = i;
    while ((j–) > 0) {
    }

    return;
}

void serialConfigNonInterrupted(unsigned char reload) {

    /* Valores úteis de reload byte:
     *            Clock (MHz)   11.0592             16.000
     * Rate (baud)
     * 2400                       0xF4          238,64 (precisa arredondar)
     * 9600                       0xFD              251,65
     *
     *
     * A fórmula e': reload = 256 - (cristal/(baud*32*12))
     *
     */

    SCON  = 0x52;   /* SCON: mode 1, 8-bit UART, enable rcvr */
    TMOD |= 0x20;   /* TMOD: timer 1, mode 2, 8-bit reload */
    TH1   = reload; /* TH1:  reload value for 9600 baud @ 11.0592MHz */
    TR1   = 1;      /* TR1:  timer 1 run */
    TI    = 1;      /* TI:   set TI to send first char of UART */
}


/* Making buzzer to sound */
void outputBuzzer() {
    if (iBuzzerTime == BUZZER_TIME) {
        /*
         * Atencao, a constante PWM_FREQUENCY denota o valor a ser carregado
         * no registrador PWMP, e nao a frequencia, que e' dada pela formula
         * Fpwm =  Fxtal / (2 * (1 + PWM_FREQUENCY) * 255)
         */

        PWMP = PWM_FREQUENCY;
        PWM0 = PWM_DUTY_CYCLE;
        PWM1 = PWM_DUTY_CYCLE;
    }

    if ((iBuzzerTime–) == 0) {
        PWM0 = 0; /* silencia o buzzer */
        PWM1 = 0; /* silencia o buzzer */
    } else {
        return;
    }
}
```

```
void init() {

    serialConfigNonInterrupted(TH1_RELOAD);

    /* IRPD global variables */
    iIrpdNWLeft = 0;
    iIrpdNWRight = 0;
    iIrpdNELeft = 0;
    iIrpdNERight = 0;
    iIrpdSWLeft = 0;
    iIrpdSWRight = 0;
    iIrpdSELeft = 0;
    iIrpdSERight = 0;

    /* line tracker variables */
    iLineTrackerLeft = 0;
    iLineTrackerCenter = 0;
    iLineTrackerRight = 0;

    /* LDR variables */
    iLDRFrontLeft = 0;
    iLDRFrontCenter = 0;
    iLDRFrontRight = 0;
    iLDRArm = 0;
    iLDRRear = 0;

    /* bumper switches variables */
    iBumperFrontLeft = 0;
    iBumperFrontRight = 0;
    iBumperLeft = 0;
    iBumperRight = 0;
    iBumperRear = 0;

    /* microphone average input level variable */
    iMicrophoneLeft = 0;
    iMicrophoneRight = 0;

    /* iBattery level sensing */
    iBattery = 0;

    /* buzzer time to sound */
    iBuzzerTime = 0;

    /* servomechanisms control */
    /*  wheel motors */
    iLeftMotor  = 127;
    iRightMotor = 127 ;

    /*  arm motors */
    iBaseServo = 127;
    iShoulderServo = 127;
    iElbowServo = 127;
    iWristServo = 127;
    iGripServo = 127;

    iBaseServoTarget = 127;
    iShoulderServoTarget = 127;
    iElbowServoTarget = 127;
    iWristServoTarget = 127;
    iGripServoTarget = 127;

    /*  motors delta values */
    iLeftMotorChanged = 0;
    iLeftMotorChanged = 0;
    iBaseServoChanged = 0;
    iShoulderServoChanged = 0;
```

```
    iElbowServoChanged = 0;
    iWristServoChanged = 0;
    iGripServoChanged = 0;

    // countdown
    P4_BITS.B6 = 0;
    printf(“.”);
    delay(DEFAULT_DELAY);
    P4_BITS.B6 = 1;
    printf(“.”);
    delay(DEFAULT_DELAY);
    P4_BITS.B6 = 0;

}


// UPDATING METHODS

void updateIRPD() {

    /**
     * For this function, the following mapping is assumed:
     * P1.5 - Left IR emitter
     * P1.6 - Right IR emitter
     * P1.7 - IR receiver
     * P4.0 - IR receiver
     * P4.1 - IR receiver
     * P4.2 - IR receiver
     */

    /**
     * Polls the Left side
     */
    P1_BITS.B5 = 1;
    P3_BITS.B3 = 0;

    delay(IR_DELAY); // lags a while before reading the port

    if (P3_BITS.B5 == 1) {
        iIrpdNWRight = 0;
    } else {
        iIrpdNWRight = 1;
    }

    if (P4_BITS.B0 == 1) {
        iIrpdNELeft = 0;
    } else {
        iIrpdNELeft = 1;
    }

    if (P4_BITS.B1 == 1) {
        iIrpdSWRight = 0;
    } else {
        iIrpdSWRight = 1;
    }

    if (P4_BITS.B2 == 1) {
        iIrpdSERight = 0;
    } else {
        iIrpdSERight = 1;
    }
```

```
    /**
     * Now polls the right side
     */
    P1_BITS.B5 = 0;
    P3_BITS.B3 = 1;
    delay(IR_DELAY); // ...

    if (P3_BITS.B5 == 1) {
        iIrpdNWLeft = 0;
    } else {
        iIrpdNWLeft = 1;
    }

    if (P4_BITS.B0 == 1) {
        iIrpdNERight = 0;
    } else {
        iIrpdNERight = 1;
    }

    if (P4_BITS.B1 == 1) {
        iIrpdSWLeft = 0;
    } else {
        iIrpdSWLeft = 1;
    }

    if (P4_BITS.B2 == 1) {
        iIrpdSELeft = 0;
    } else {
        iIrpdSELeft = 1;
    }

    P1_BITS.B5 = 0;
    P3_BITS.B3 = 0;

    return;
}



/* line tracker variable updates */
void processTrackerInputs() {
    /*
     * The following mapping is assumed
     * P4.3     -   Left   Rx
     * P4.4     -   Right  Rx
     * P4.5     -   Center Rx
     */

    if (P4_BITS.B3 == 1) {
        iLineTrackerRight = 1;
    } else {
        iLineTrackerRight = 0;
    }

    if (P4_BITS.B5 == 1) {
        iLineTrackerLeft = 1;
    } else {
        iLineTrackerLeft = 0;
    }
```

```
    if (P4_BITS.B4 == 1) {
        iLineTrackerCenter = 1;
    } else {
        iLineTrackerCenter = 0;
    }

    return;
}


/* A/D converter ports inputs */
void ADPollConverter() {

    /* faz o poll do pino 0 */
    ADCI = 0; // limpa o indicador de final de conversao A/D
    // ADCON = (ADCON & 0xf8); ?

    // Pino 0
    ADCON_BITS.B2 = 0;
    ADCON_BITS.B1 = 0;
    ADCON_BITS.B0 = 0;

    ADCS = 1; // inicia a conversao A/D
    asm(“ nop”);
    asm(“ nop”);

    while (ADCI != 1) {  // espera a conversao terminar
    }

    iLDRFrontLeft  = ADCH; // salva o resultado da conversao

    /* Faz o poll do pino 1  */
    ADCI = 0; // limpa o indicador de final de conversao A/D
    //ADCON = (ADCON & 0xf8); ?

    // Pino 1
    ADCON_BITS.B2 = 0;
    ADCON_BITS.B1 = 0;
    ADCON_BITS.B0 = 1;

    ADCS = 1; // Inicia a conversao A/D
    asm(“ nop”);
    asm(“ nop”);

    while (ADCI == 0) { // Espera a conversao terminar
    }

    iLDRFrontCenter = ADCH; // Salva o resultado da conversao

    /* Faz o poll do pino 2  */
    ADCI = 0; // Limpa o indicador de final de conversao A/D
    // ADCON = (ADCON & 0xf8); ?

    // Pino 2
    ADCON_BITS.B2 = 0;
    ADCON_BITS.B1 = 1;
    ADCON_BITS.B0 = 0;

    ADCS = 1; // Inicia a conversao A/D
    asm(“ nop”);
    asm(“ nop”);

    while (ADCI == 0) { // Espera a conversao terminar
    }

    iLDRFrontRight = ADCH; // Salva o resultado da conversao
```

```
/* Faz o poll do pino 3  */
ADCI = 0; // Limpa o indicador de final de conversao A/D
// ADCON = (ADCON & 0xf8); ?

// Pino 3
ADCON_BITS.B2 = 0;
ADCON_BITS.B1 = 1;
ADCON_BITS.B0 = 1;

ADCS = 1; // Inicia a conversao A/D
asm(“ nop”);
asm(“ nop”);

while (ADCI == 0) { // Espera a conversao terminar
}

iLDRArm = ADCH; // Salva o resultado da conversao


/* Faz o poll do pino 4  */
ADCI = 0; // Limpa o indicador de final de conversao A/D
// ADCON = (ADCON & 0xf8); ?

// Pino 4
ADCON_BITS.B2 = 1;
ADCON_BITS.B1 = 0;
ADCON_BITS.B0 = 0;

ADCS = 1; // Inicia a conversao A/D
asm(“ nop”);
asm(“ nop”);

while (ADCI == 0) { // Espera a conversao terminar
}

iLDRRear = ADCH; // Salva o resultado da conversao


/* Faz o poll do pino 5  */
ADCI = 0; // Limpa o indicador de final de conversao A/D
// ADCON = (ADCON & 0xf8); ?

// Pino 5
ADCON_BITS.B2 = 1;
ADCON_BITS.B1 = 0;
ADCON_BITS.B0 = 1;

ADCS = 1; // Inicia a conversao A/D
asm(“ nop”);
asm(“ nop”);

while (ADCI == 0) { // Espera a conversao terminar
}

iMicrophoneLeft = ADCH; // Salva o resultado da conversao


/* Faz o poll do pino 6  */
ADCI = 0; // Limpa o indicador de final de conversao A/D
// ADCON = (ADCON & 0xf8); ?
```

```
    // Pino 6
    ADCON_BITS.B2 = 1;
    ADCON_BITS.B1 = 1;
    ADCON_BITS.B0 = 0;

    ADCS = 1; // Inicia a conversao A/D
    asm(“ nop”);
    asm(“ nop”);

    while (ADCI == 0) { // Espera a conversao terminar
    }

    iMicrophoneRight = ADCH; // Salva o resultado da conversao


    /* Faz o poll do pino 7  */
    ADCI = 0; // Limpa o indicador de final de conversao A/D
    // ADCON = (ADCON & 0xf8); ?

    // Pino 7
    ADCON_BITS.B2 = 1;
    ADCON_BITS.B1 = 1;
    ADCON_BITS.B0 = 1;

    ADCS = 1; // Inicia a conversao A/D
    asm(“ nop”);
    asm(“ nop”);

    while (ADCI == 0) { // Espera a conversao terminar
    }

    iBattery = ADCH; // Salva o resultado da conversao

    ADCI = 0;

    return;
}


/* Bumper switches polling */

void BumperPoll() {

    if (P1_BITS.B0 == 1) {
        iBumperFrontLeft = 1;
    } else {
        iBumperFrontLeft = 0;
    }

    if (P1_BITS.B1 == 1) {
        iBumperFrontRight = 1;
    } else {
        iBumperFrontRight = 0;
    }

    if (P1_BITS.B2 == 1) {
        iBumperLeft = 1;
    } else {
        iBumperLeft = 0;
    }

    if (P1_BITS.B3 == 1) {
        iBumperRight = 1;
    } else {
        iBumperRight = 0;
    }
```

```
    if (P1_BITS.B4 == 1) {
        iBumperRear = 1;
    } else {
        iBumperRear = 0;
    }

    return;
}


void sendToSSC(unsigned char motor, unsigned char value) {

    unsigned char Comando[4];

    Comando[0] = 0xff;
    Comando[1] = motor;
    Comando[2] = value;
    Comando[3] = '\0';

    printf("%s", Comando);
    return;
}


void computeMotorDelta() {

    // right motor routines
    if (iRightMotorChanged == 1) {
        sendToSSC(RIGHT_MOTOR, iRightMotor);
        iRightMotorChanged = 0;
    }

    // left motor routines
    if (iLeftMotorChanged == 1) {
        sendToSSC(LEFT_MOTOR, iLeftMotor);
        iLeftMotorChanged = 0;
    }

    // base servo routines
    if (iBaseServoTarget == iBaseServo) {
        iBaseServoChanged = 0;
    } else if (iBaseServoTarget > iBaseServo) {
        iBaseServo+=MOTOR_STEP;
        iBaseServoChanged = 1;
    } else if (iBaseServoTarget < iBaseServo) {
        iBaseServo-=MOTOR_STEP;
        iBaseServoChanged = 1;
    }

    if (iBaseServoChanged == 1) {
        sendToSSC(BASE_MOTOR, iBaseServo);
        iBaseServoChanged = 0;
    }

    // shoulder servo routines
    if (iShoulderServoTarget == iShoulderServo) {
        iShoulderServoChanged = 0;
    } else if (iShoulderServoTarget > iShoulderServo) {
        iShoulderServo+=MOTOR_STEP;
        iShoulderServoChanged = 1;
    } else if (iShoulderServoTarget < iShoulderServo) {
        iShoulderServo-=MOTOR_STEP;
        iShoulderServoChanged = 1;
    }
```

```
    if (iShoulderServoChanged == 1) {
        sendToSSC(SHOULDER_MOTOR, iShoulderServo);
        iShoulderServoChanged = 0;
    }

    // elbow servo routines
    if (iElbowServoTarget == iElbowServo) {
        iElbowServoChanged = 0;
    } else if (iElbowServoTarget > iElbowServo) {
        iElbowServo+=MOTOR_STEP;
        iElbowServoChanged = 1;
    } else if (iElbowServoTarget < iElbowServo) {
        iElbowServo-=MOTOR_STEP;
        iElbowServoChanged = 1;
    }

    if (iElbowServoChanged == 1) {
        sendToSSC(ELBOW_MOTOR, iElbowServo);
        iElbowServoChanged = 0;
    }

    // wrist servo routines
    if (iWristServoTarget == iWristServo) {
        iWristServoChanged = 0;
    } else if (iWristServoTarget > iWristServo) {
        iWristServo+=MOTOR_STEP;
        iWristServoChanged = 1;
    } else if (iWristServoTarget < iWristServo) {
        iWristServo-=MOTOR_STEP;
        iWristServoChanged = 1;
    }

    if (iWristServoChanged == 1) {
        sendToSSC(WRIST_MOTOR, iWristServo);
        iWristServoChanged = 0;
    }

    // grip servo routines
    if (iGripServoTarget == iGripServo) {
        iGripServoChanged = 0;
    } else if (iGripServoTarget > iGripServo) {
        iGripServo+=MOTOR_STEP;
        iGripServoChanged = 1;
    } else if (iGripServoTarget < iGripServo) {
        iGripServo-=MOTOR_STEP;
        iGripServoChanged = 1;
    }

    if (iGripServoChanged == 1) {
        sendToSSC(GRIP_MOTOR, iGripServo);
        iGripServoChanged = 0;
    }

    return;
}
```

```
void update() {

    // P4_BITS.B6 = !P4_BITS.B6; ?

    // processa IRs
    updateIRPD();

    // processa trackers
    processTrackerInputs();

    // processa mic, LDRs e baterias
    ADPollConverter();

    // processa chaves de contato
    BumperPoll();

    // gerencia a saída do Buzzer
    outputBuzzer();

    // manda saída para motores
    computeMotorDelta();

    Follow_counter = (Follow_counter > 0 ? Follow_counter - 1 : 0);
    Avoid_counter = (Avoid_counter > 0 ? Avoid_counter - 1 : 0);
    Escape_counter = (Escape_counter > 0 ? Escape_counter - 1 : 0);

    return;
}


// Comentar para reduzir tamanho do programa
/*
printAllInFormattedWay() {

    printf("\n\n");

    printf("IRPD: [ %d %d ][ %d %d ][ %d %d ][ %d %d ]\n",
        iIrpdNWLeft, iIrpdNWRight, iIrpdNELeft, iIrpdNERight,
        iIrpdSWLeft, iIrpdSWRight,iIrpdSELeft, iIrpdSERight);

    printf("Trackers: [%d %d %d ] \n",
        iLineTrackerLeft, iLineTrackerCenter, iLineTrackerRight);

    printf("AD: %x %x %x %x %x %x %x %x\n",
        iLDRFrontLeft, iLDRFrontCenter, iLDRFrontRight, iLDRArm, iLDRRear,
        iMicrophoneLeft, iMicrophoneRight, iBattery);

    printf("Bumper: %d %d %d %d %d\n", iBumperFrontLeft,
        iBumperFrontRight, iBumperLeft, iBumperRight, iBumperRear);

    printf("Trying  base motor");

    // sendToSSC(BASE_MOTOR, 140); ?

    return;
} */

/*
void main(void){ // utilizada apenas em caso de debugação

    init();
    setBuzzer();
    while(1){
        update();
        printAllInFormattedWay();
```

```
        delay(DEFAULT_DELAY);
    }
}
*/

#define SHOWMSG 1
/**
 * The compiled behavior
 */


// ——————————————————
// the processes
// ——————————————————

void Fred(void) {

    if (SHOWMSG) printf("Fred %c, ", (char) (Fred_next + 48));

    // code before
    // set values from inputs
    setLeftMotor(Fred_LeftMotor);
    setRightMotor(Fred_RightMotor);
    if (Fred_Buzzer) {
        setBuzzer();
    }
    // the most important call in this process
    update();
    // copy values to outputs
    Fred_IrpdNWLeft = getIrpdNWLeft();
    Fred_IrpdNWRight = getIrpdNWRight();
    Fred_IrpdNELeft = getIrpdNELeft();
    Fred_IrpdNERight = getIrpdNERight();
    Fred_LineTrackerLeft = getLineTrackerLeft();
    Fred_LineTrackerCenter = getLineTrackerCenter();
    Fred_LineTrackerRight = getLineTrackerRight();
    Fred_LDRFrontLeft = getLDRFrontLeft();
    Fred_LDRFrontCenter = getLDRFrontCenter();
    Fred_LDRFrontRight = getLDRFrontRight();
    Fred_LDRArm = getLDRArm();
    Fred_BumperFrontLeft = getBumperFrontLeft();
    Fred_BumperFrontRight = getBumperFrontRight();
    Fred_BumperLeft = getBumperLeft();
    Fred_BumperRight = getBumperRight();
    Fred_BumperRear = getBumperRear();
    Fred_MicrophoneLeft = getMicrophoneLeft();

    // sets "talk to" flag
    Fred_To = 1;

    return;
}

// ——————————————————

void Cruise(void) {

    if (SHOWMSG) printf("Cruise %c, ", (char) (Cruise_next + 48));
    // code before
    Cruise_LeftMotor= 110;
    Cruise_RightMotor= 110;

    // sets "talk to" flag
    Cruise_To = 1;

    return;
}
```

```
// ————————————————

void Follow(void) {

    if (SHOWMSG) printf("Follow %c, ", (char) (Follow_next + 48));

    // code before
    // at least one has a minimum light
    Follow_trigger = (Follow_LDRFrontLeft > 150 ||
    Follow_LDRFrontCenter > 150 ||
    Follow_LDRFrontRight > 150);
    // calculates the balance
    Follow_deltaLeft = (Follow_LDRFrontLeft - Follow_LDRFrontCenter);
    Follow_deltaRight = (Follow_LDRFrontRight - Follow_LDRFrontCenter);
    // if the light is outside the center position
    Follow_trigger = Follow_trigger && (Follow_deltaLeft >0 || Follow_deltaRight > 0);

    // sets "talk to" flag
    Follow_To = 0;

    // next state logic
    switch (Follow_next) {

        case 1:
            if (Follow_trigger) {
                Follow_counter = 5;
                Follow_next = 2;
            }
        break;

        case 2:
            // turn to the left
            if (Follow_deltaLeft > Follow_deltaRight) {
                // make an arc
                Follow_RightMotor = 80;
                Follow_LeftMotor = 100;
                // turn to the right
            } else if (Follow_deltaLeft > Follow_deltaRight) {
                // make an arc
                Follow_RightMotor = 100;
                Follow_LeftMotor = 80;
            }
            if (Follow_counter— == 0) {
                Follow_next = 1;
            }
            Follow_To = 1;
        break;
    }

    return;
}

// ————————————————

void Avoid(void) {


    // states numbers
    enum { stateChecking,
           statePanic,
           stateHardLeft,
           stateSoftLeft,
           stateHardRight,
           stateSoftRight
    };
```

```
if (SHOWMSG) printf("Avoid %c", (char) (Avoid_next + 48));
// clear flag
Avoid_To = 0;

// next state logic
switch (Avoid_next) {

    case stateChecking:
        Avoid_timer = 20;
        // if there is something...
        if (Avoid_IrpdNERight || Avoid_IrpdNELeft ||
        Avoid_IrpdNWRight || Avoid_IrpdNWLeft) {
            // if it's both sides, turn away
            if ((Avoid_IrpdNERight || Avoid_IrpdNELeft) &&
            (Avoid_IrpdNWRight || Avoid_IrpdNWLeft)) {
                Avoid_next = statePanic;
            } else {
                // if it's only in the NE
                if (Avoid_IrpdNERight || Avoid_IrpdNELeft) {
                    // if it needs a hard turn
                    if (Avoid_IrpdNELeft) {
                        Avoid_next = stateHardLeft;
                    } else {
                        Avoid_next = stateSoftLeft;
                    }
                    // if it's only in the NW
                } else {
                    // if it needs a hard turn
                    if (Avoid_IrpdNWRight) {
                        Avoid_next = stateHardRight;
                    } else {
                        Avoid_next = stateSoftRight;
                    }
                }
            }
        }
        // spins to the left
    break;

    case statePanic:
        Avoid_RightMotor = 77;
        Avoid_LeftMotor = 177;
        if (Avoid_timer– <= 0) {
            Avoid_next = stateChecking;
        }

        Avoid_To = 1;
    break;

    case stateHardLeft:
        Avoid_RightMotor = 50;
        Avoid_LeftMotor = 80;
        if (Avoid_timer– <= 0) {
            Avoid_next = stateChecking;
        }

        Avoid_To = 1;
    break;
```

```
        case stateSoftLeft:
            Avoid_RightMotor = 80;
            Avoid_LeftMotor = 100;
            if (Avoid_timer— <= 0) {
                Avoid_next = stateChecking;
            }

            Avoid_To = 1;
        break;

        case stateHardRight:
            Avoid_RightMotor = 80;
            Avoid_LeftMotor = 50;
            if (Avoid_timer— <= 0) {
                Avoid_next = stateChecking;
            }

            Avoid_To = 1;
        break;


        case stateSoftRight:
            Avoid_RightMotor = 100;
            Avoid_LeftMotor = 80;
            if (Avoid_timer— <= 0) {
                Avoid_next = stateChecking;
            }

            Avoid_To = 1;
        break;

    }

    return;
}

// ——————————————

void Escape(void) {


    // states numbers
    enum { stateChecking,
           statePanic,
           statePanicSolved,
           stateFrontLeft,
           stateFrontRight,
           stateRight,
           stateLeft,
           stateRear,
           stateTryAgain
    };


    if (SHOWMSG) printf("Escape %c", (char) (Escape_next + 48));
    // clear flag
    Escape_To = 0;
```

```
    // next state logic
    switch (Escape_next) {

        case stateChecking:
            Escape_timer = 10;
            // if either in the front
            if (Escape_BumperFrontLeft || Escape_BumperFrontRight) {
                if (Escape_BumperFrontLeft && Escape_BumperFrontRight) {
                    Escape_timer = 20;
                    Escape_next = statePanic;
                } else if (Escape_BumperFrontLeft) {
                    Escape_next = stateFrontLeft;
                } else {
                    Escape_next = stateFrontRight;
                }
            } else if (Escape_BumperRight) {
                Escape_next = stateRight;
            } else if (Escape_BumperLeft) {
                Escape_next = stateLeft;
            } else if (Escape_BumperRear) {
                Escape_next = stateRear;
            }
        break;

        case statePanic:
            // first, just go back for a while
            Escape_RightMotor = 140;
            Escape_LeftMotor = 140;
            if (Escape_timer_ <= 0) {
                Escape_timer = 15;
                Escape_next = statePanicSolved;
            }

            Escape_To = 1;
        break;

        case statePanicSolved:
            Escape_RightMotor = 147;
            Escape_LeftMotor = 107;
            if (Escape_timer_ <= 0) {
                Escape_timer = 10;
                Escape_next = stateTryAgain;
            }

            Escape_To = 1;
        break;

        case stateFrontLeft:
            Escape_RightMotor = 140;
            Escape_LeftMotor = 160;
            if (Escape_timer_ <= 0) {
                Escape_timer = 10;
                Escape_next = stateTryAgain;
            }

            Escape_To = 1;
        break;

        case stateFrontRight:
            Escape_RightMotor = 160;
            Escape_LeftMotor = 140;
            if (Escape_timer_ <= 0) {
                Escape_timer = 10;
                Escape_next = stateTryAgain;
            }

            Escape_To = 1;
        break;
```

```
        case stateRight:
            Escape_RightMotor = 140;
            Escape_LeftMotor = 127;
            if (Escape_timer— <= 0) {
                Escape_timer = 10;
                Escape_next = stateTryAgain;
            }

            Escape_To = 1;
        break;

        case stateLeft:
            Escape_RightMotor = 127;
            Escape_LeftMotor = 140;
            if (Escape_timer— <= 0) {
                Escape_timer = 10;
                Escape_next = stateTryAgain;
            }

            Escape_To = 1;
        break;

        case stateRear:
            Escape_RightMotor = 100;
            Escape_LeftMotor = 120;
            if (Escape_timer— <= 0) {
                Escape_timer = 10;
                Escape_next = stateTryAgain;
            }

            Escape_To = 1;
        break;

        case stateTryAgain:
            Escape_RightMotor = 117;
            Escape_LeftMotor = 117;
            if (Escape_timer— <= 0) {
                Escape_next = stateChecking;
            }

            Escape_To = 1;
        break;

    }

    return;
}
```

```
// ————————————————

void Connections(void) {

    // priority 0
    if (Cruise_To) {
        Fred_LeftMotor = Cruise_LeftMotor;
        Fred_RightMotor = Cruise_RightMotor;
    }

    // priority 1
    if (Fred_To) {
        Follow_LDRFrontLeft = Fred_LDRFrontLeft;
        Follow_LDRFrontCenter = Fred_LDRFrontCenter;
        Follow_LDRFrontRight = Fred_LDRFrontRight;
    }
    if (Follow_To) {
        Fred_LeftMotor = Follow_LeftMotor;
        Fred_RightMotor = Follow_RightMotor;
    }

    // priority 2
    if (Fred_To) {
        Avoid_IrpdNWLeft = Fred_IrpdNWLeft;
        Avoid_IrpdNWRight = Fred_IrpdNWRight;
        Avoid_IrpdNELeft = Fred_IrpdNELeft;
        Avoid_IrpdNERight = Fred_IrpdNERight;
    }
    if (Avoid_To) {
        Fred_LeftMotor = Avoid_LeftMotor;
        Fred_RightMotor = Avoid_RightMotor;
    }

    // priority 3
    if (Fred_To) {
        Escape_BumperFrontLeft = Fred_BumperFrontLeft;
        Escape_BumperFrontRight = Fred_BumperRight;
        Escape_BumperLeft = Fred_BumperLeft;
        Escape_BumperRight = Fred_BumperRight;
        Escape_BumperRear = Fred_BumperRear;
    }

    // priority 4
    if (Escape_To) {
        Fred_RightMotor = Escape_RightMotor;
        Fred_LeftMotor = Escape_LeftMotor;
    }

    return;
}
```

```
// ——————————————————

void main(void) {

    // processes counter
    int counter;

    // initialize vars
    counter = 0;
    Fred_next = 1;
    Cruise_next = 1;
    Follow_next = 1;
    Avoid_next = 1;
    Escape_next = 1;
    Fred_LeftMotor = 123;
    Fred_RightMotor = 123;

    // initialize driver
    init();

    // program loop (forever)
    do {
        counter++;

        if (counter % 1 == 0) {
            Fred();
            Connections();
        }

        if (counter % 5 == 0) {
            Cruise();
            Connections();
        }

        if (counter % 4 == 0) {
            Follow();
            Connections();
        }

        if (counter % 2 == 0) {
            Avoid();
            Connections();
        }

        if (counter % 1 == 0) {
            Escape();
            Connections();
        }

    } while(1);

    return;
}
```

Patrocinadores:

Departamento de Engenharia de Computação e Sistemas Digitais

Escola Politécnica

Universidade de São Paulo

ANACOM SOFTWARE