Pontifícia Universidade Católica
do Rio de Janeiro

Fabiano Correia Santério

Controle baseado em comportamentos de robôs móveis autônomos com sensores ópticos e ultrassônicos

Dissertação de Mestrado

Dissertação apresentada como requisito parcial para obtenção do título de Mestre pelo Programa de Pós-Graduação em Engenharia Mecânica da PUC-Rio.

Orientador: Prof. Marco Antonio Meggiolaro

Rio de Janeiro, 5 de março de 2010

PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO

Fabiano Correia Santério

Controle baseado em comportamentos de robôs móveis autônomos com sensores ópticos e ultrassônicos

Dissertação apresentada como requisito parcial para obtenção do título de Mestre pelo Programa de Pós-Graduação em Engenharia Mecânica da PUC-Rio. Aprovada pela Comissão Examinadora abaixo assinada.

Prof. Marco Antonio Meggiolaro
Orientador
Departamento de Engenharia Mecânica - PUC-Rio

Prof. Mauro Speranza Neto
Departamento de Engenharia Mecânica - PUC-Rio

Prof. Armando Morado Ferreira
Instituto Militar de Engenharia - IME

Prof. José Eugênio Leal
Coordenador(a) Setorial do Centro
Técnico Científico - PUC-Rio

Rio de Janeiro, 5 de Março de 2010

Fabiano Correia Santério

Graduou-se em Engenharia de Controle e Automação (Pontifícia Universidade Católica) em 2005. Trabalha a quatro anos na área de automação predial e naval nos setores de projeto, implementação e controle de sistemas. Iniciou o mestrado na área de Mecânica Aplicada na Pós-Graduação da PUC-Rio em 2007. Suas áreas de interesse abrangem projeto, implementação e controle processos de automação industrial, naval e predial assim como na área da robótica voltada para indústria *offshore*.

Ficha Catalográfica

Santério, Fabiano Correia

Controle baseado em comportamentos de robôs móveis autônomos com sensores ópticos e ultrassônicos / Fabiano Correia Santério ; orientador: Marco Antonio Meggiolaro. – 2010.

147 f. : il. (color.) ; 30 cm

Dissertação (Mestrado em Engenharia Mecânica)–Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2010.

Inclui bibliografia

1. Engenharia mecânica – Teses. 2. Robô móvel. 3. Robô autônomo. 4. Controle baseado em comportamentos. 5. Sensores ópticos. 6. Sensores ultrassônicos. I. Meggiolaro, Marco Antonio. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Engenharia Mecânica. III. Título.

CDD: 621

aos meus pais, Edison e Laura.

Agradecimentos

À Deus.

À Vice-Reitoria Acadêmica (CCPG), pelo apoio financeiro;

À PUC-Rio pelo seu excelente corpo docente e infra-estrutura que possibilitou toda minha plataforma de estudos;

Ao orientador Marco Antonio Meggiolaro, pela dedicação, ensinamentos, orientação e incentivos que me fizeram alcançar meus objetivos;

Ao professor Mauro Speranza Neto, pelo apoio e orientação que me guiaram e incentivaram desde a época da graduação;

A meu pai e minha mãe, que sempre me deixaram livre nas escolhas de meu futuro e ao mesmo tempo apoiaram e incentivaram todas minhas decisões acadêmicas e profissionais baseados na ética e valores morais que me passaram;

Aos alunos de graduação e pós-graduação que frequentaram o laboratório de controle e automação durante todo meu mestrado, que me ajudaram muito para a conclusão desse trabalho;

Resumo

Santério, Fabiano C., Meggiolaro, Marco A. **Controle baseado em comportamentos de robôs móveis autônomos com sensores ópticos e ultrassônicos.** Rio de Janeiro 2010. 147p. Dissertação de Mestrado - Departamento de Engenharia Mecânica, Pontifícia Universidade Católica do Rio de Janeiro.

O comportamento animal serviu de inspiração para o controle baseado por comportamento aplicado a robôs móveis autônomos. Este se baseia em diretivas simples, ou reações, separadas por camadas com respectivas prioridades. Quando agrupadas, estas diretivas conseguem executar as mais diversas e complexas funções no ambiente, tornando o controle em si uma tarefa segmentada, na qual se divide o objetivo principal em pequenos módulos chamados de comportamentos primários. Estes atuam independentes e, ao trabalharem de forma paralela resultam em comportamentos complexos capazes de realizar tarefas mais complexas, uma evolução do controle reativo. A lógica desta técnica torna mais simples a programação e organização das tarefas porque possui uma estrutura modular que permite a adição de novos sensores (novos comportamentos), sem grandes mudanças no código existente. Esta dissertação desenvolve e implementa a programação baseada em comportamento em robôs móveis autônomos com sensores óticos e ultrassônicos em um ambiente de simulação *open source* muito comum chamado Player/Stage e validada experimentalmente em 2 robôs autônomos. A arquitetura utilizada no processo de desenvolvimento das camadas comportamentais foi a arquitetura de esquemas motores em conjunto com a técnica de campos potenciais, originalmente idealizada por Ronald C. Arkin em 1998. Os resultados simulados e experimentais foram confrontados com métodos de programação clássica e comprovam todas as vantagens do controle baseado em comportamento.

Palavras-chave

Robô móvel; robô autônomo; controle baseado em comportamentos; sensores ópticos; sensores ultrassônicos.

Abstract

Santério, Fabiano C., Meggiolaro, Marco A. (Advisor) **Behavior based control of autonomous robots with optical and ultrasonic sensors.** Rio de Janeiro 2010. 147p. MSc Dissertation – Departamento de Engenharia Mecânica, Pontifícia Universidade Católica do Rio de Janeiro.

The animal behavior served as the inspiration for the behavior based control applied to autonomous mobile robots. This is based on simple directives, or reactions, separated by layers with their priorities. When combined, these directives can run the most diverse and complex functions in the environment, making the control itself a segmented task, which divides the main objective into small modules called primary behaviors. Those act independently and, when working in parallel, result in complex behaviors that are capable of executing more complex tasks, an evolution of reactive control. The logic from this technique makes it easy to program and organize robot controllers because it has a modular structure that allows the addition of new sensors (new behaviors) without major changes in the existing code. This thesis develops and implements a behavior based programming on autonomous mobile robots with optical and ultrasonic sensors in very common open source simulation software, called Player/Stage, and validates experimentally in 2 autonomous robots. The architecture used in the development of behavioral layers was the motor schema with potential fields, originally created by Ronald C. Arkin in 1998. The simulated and experimental results were confronted with classical methods of programming and proved all the benefits of behavior based control.

Keywords

Mobile robot; autonomous robot; behavior based control; optical sensors; ultrasonic sensors.

Sumário

Lista de figuras

1 Introdução

1.1. Origens

A biologia trouxe através dos anos inúmeras formas de controle primitivo, que num contexto ambiental geram resultados complexos e eficientes.

A inspiração para o controle baseado em comportamento aplicado a robôs móveis autônomos veio do comportamento animal. Trata-se de uma metodologia baseada em um conjunto de módulos interativos, ou comportamentos primitivos, que são processos ou leis de controle que possuem um objetivo bem definido por exemplo, o de 'desviar de obstáculos', que traduz a idéia de evitar colisões e 'buscar', representando a idéia de procurar um objeto ou localidade.

Nos anos 80 foram dados os primeiros passos no desenvolvimento desta metodologia: o professor Rodney Brooks (1986) desenvolveu e implementou a arquitetura de subsunção em robôs autônomos.

O controle baseado em comportamento é fundamentalmente baseado em reações. Não há um plano de ações a serem executadas e não existe um modelo ambiental pré-definido, ou seja, toda a informação é obtida através do sensoriamento existente. O robô percebe e reage ao ambiente e suas variações à medida que se locomove, o que reforça a definição de Brooks (BROOKS 1986) [16]: "Planejar é apenas uma maneira de evitar descobrir o que fazer a seguir" e de Arkin (ARKIN, 1998) [1] que diz, "Percepção é uma impressão obtida de um objeto pelo uso de sentidos: *sense-datum*". Pode-se entender a impressão de vários objetos como a modelagem do ambiente desejado utilizando sensoriamento diverso.

Um observador externo interpreta os comportamentos como padrões de atividade do robô emergentes das interações entre o robô e seu ambiente, já um programador vê os comportamentos como módulos de controle com funções bem definidas e limitadas que trabalham em conjunto a fim de alcançar uma meta desejada.

Cada comportamento pode receber entradas de sensores como câmeras, ultrassom, infravermelho e/ou outros comportamentos no sistema e oferece saídas para os atuadores do robô, como rodas, garras, braços e/ou outros comportamentos. Quando agrupados, eles conseguem executar as mais diversas e complexas funções no ambiente, tornando o controle em si uma tarefa segmentada. Esta tarefa é a divisão do objetivo principal em pequenos módulos, que de uma forma ideal atuam independentemente uns dos outros, mas que em conjunto resultam em comportamentos complexos.

Os comportamentos primários possuem um estado de atuação que, agrupados de maneira a serem executados paralelamente, podem criar representações que, ao contrário dos sistemas puramente reativos, não são limitadas nas capacidades de expressão e aprendizado.

A lógica derivada desta técnica torna mais simples a programação e organização das tarefas pois, com sua estrutura modular, permite inclusive a adição de novos sensores sem grandes mudanças no código já existente. Nesse caso, os sensores adicionais proveriam mais "comportamentos" que atuariam em algum nível da hierarquia de controle.

1.2. Motivação

Os sistemas móveis autônomos atuais enfrentam dois grandes problemas em sua arquitetura: a localização relativa ao ambiente com precisão suficiente para a tarefa e a complexidade da programação das ações a serem executadas, face à grande quantidade de sensores. Estes sensores geram inúmeras combinações de atitudes possíveis, as quais precisam ser planejadas e calculadas antes do processo se iniciar.

Tarefas a serem executadas por robôs móveis autônomos em ambientes desconhecidos geram um grande problema em relação à auto-localização e à extensa programação agregada ao processo, pois o mapeamento do ambiente de trabalho utiliza técnicas sofisticadas que necessitam de um alto grau de processamento, acarretando numa alta complexidade em sua lógica.

Robôs autônomos interplanetários são um exemplo de alta complexidade de programação, pois necessitam de muitos sensores com finalidades diferentes que

proporcionam informações variadas, gerando uma extensa e complexa programação das tarefas. (Figura 1).

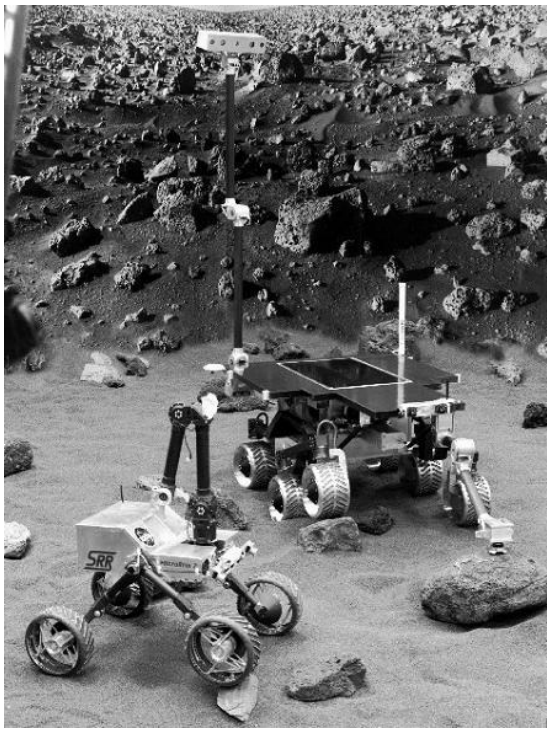


Figura 1 - Robôs interplanetários SRR (Sample Return Rover) e FIDO (Field Integrated Design and Operations). (SCHENKER, 2003) [14]

Uma das soluções para o problema do alto grau de complexidade de programação seria utilizar a técnica de controle reativo (Figura 2), que se baseia em determinar reações a serem tomadas de acordo com a leitura dos sensores. Este processo é simples do ponto de vista computacional, pois pode ser aplicada a cada sensor independentemente gerando respostas rápidas ao sistema. Este tipo de controle resulta em baixos tempos de resposta, porém com um grau de acerto baixo devido a não haver hierarquias entre os sensores.

Figura 2 - Controle Reativo em robô com funções de escritório. (FENG, 1994) [9]

O controle baseado por comportamento agrega valor ao controle reativo, pois cria módulos básicos de programação das reações aos dados de entrada e define uma hierarquia e prioridades entre eles. Brooks (BROOKS, 1999) [16] desenvolveu Genghis I, um robô inseto de seis patas utilizando o controle baseado por comportamento. (Figura 3).

Figura 3 - Robô inseto Genghis I controlado por comportamento. (BROOKS, 1999) [16]

Esta teoria serviu como base para o desenvolvimento de um controle de múltiplos robôs em um mesmo ambiente, com os mesmos comportamentos, que agregados geram resultados eficientes e com muita robustez.

Goldberg (GOLDBERG, 2001) [45] criou um sistema de controle baseado em comportamento de quatro robôs R2e com objetivo de coletar material tóxico e que interajam entre si de forma a completar a tarefa da maneira mais eficiente possível e com a robustez de execução mesmo que um ou mais robôs apresentem algum defeito. (Figura 4).

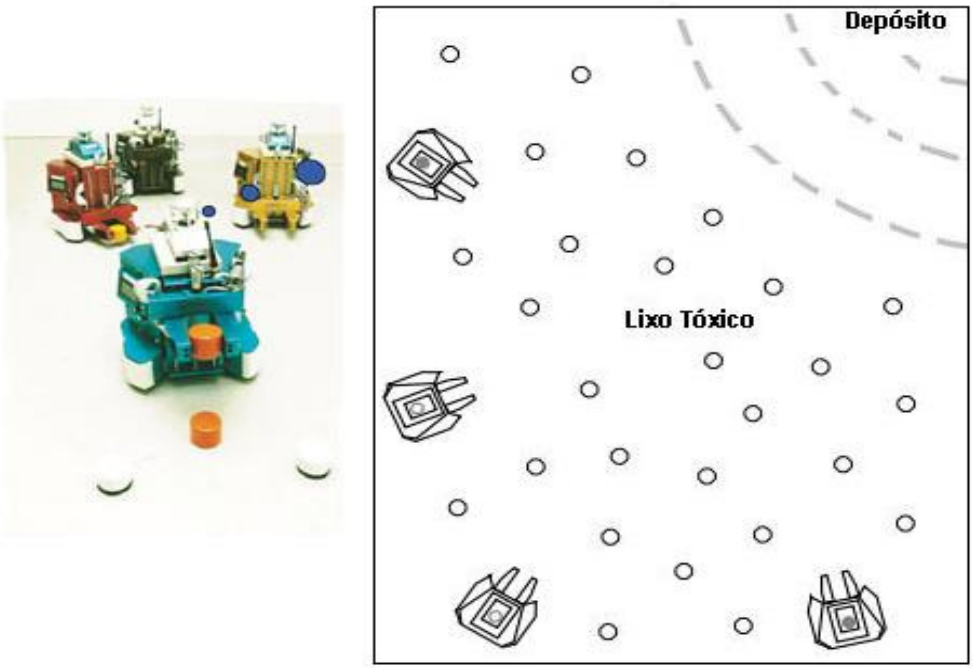


Figura 4 - Robôs R2e baseados em comportamento para coleta de lixo tóxico. (GOLDBERG, 2001) [45]

O controle baseado em comportamento utilizando um sensoriamento ótico incremental de alta resolução e conjunto com sensores ultrassônicos e infravermelhos é proposto neste trabalho para obter um sistema facilmente programável e com rápido tempo de resposta.

Outra grande aplicação que utiliza o controle comportamental cooperativo é o futebol de robôs. Existem várias categorias desta modalidade, mas a que mais se destaca é a que utiliza os robôs com quatro membros articulados com intuito de simular um cachorro, chamado AIBO da empresa *SONY* (Figura 5). Peter Stones (STONE 2007) [46] apresenta todo o processo de estudo e implementação da teoria do controle por comportamento para o futebol de robôs. (Figura 6).

Figura 5 - Robôs AIBO da empresa SONY. Simula o comportamento de um cachorro real. (STONE, 2007) [46]

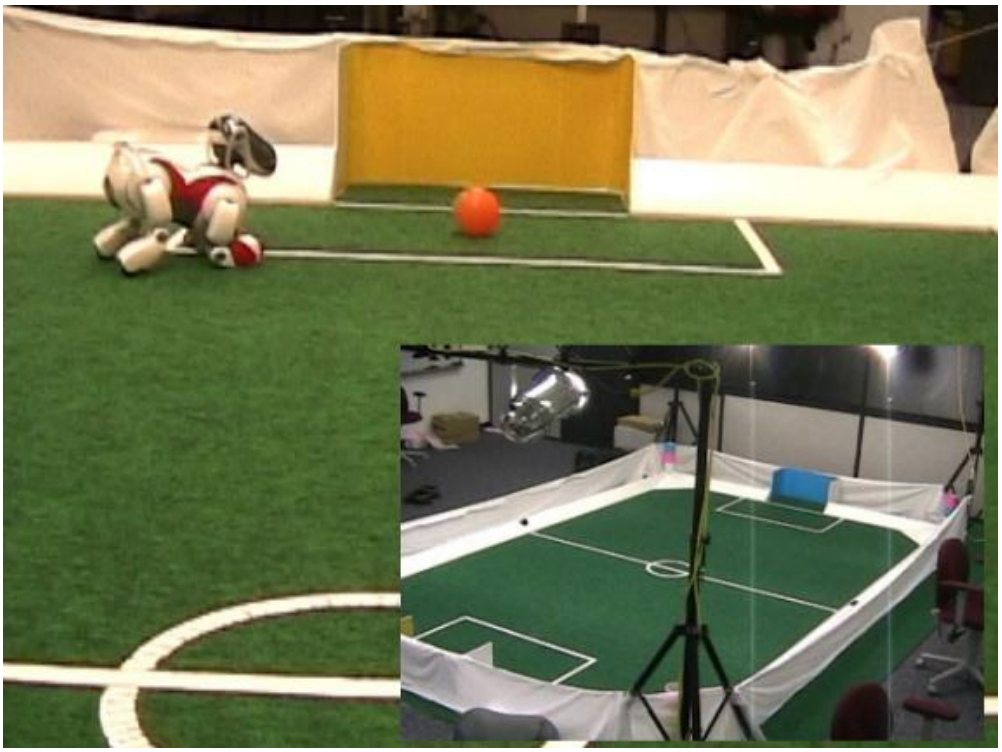


Figura 6 – Campo de competição de futebol de robôs. (STONE, 2007) [46]

Mataric (MATARIC, 1997) [47] publicou diversos trabalhos que abordam o controle baseado em comportamento para robôs móveis autônomos. Desenvolveu projetos para robôs representarem individualmente ou em grupo comportamentos exploratórios, ataque, dispersão, formação em cadeia dentre outros. (Figura 7)

Figura 7 – Robôs ISX utilizados para experimentos com controle baseado em comportamento. (MATARIC, 1997) [47]

Aliada ao controle baseado em comportamento, a técnica de campos potenciais inicialmente implementada por Khatib (KHATIB, 1985) [21], Andrews (ANDREWS, 1983) [22] e Krogh (KROGH, 1984) [23] traz grande contribuição na navegação do dispositivo, pois trata o conteúdo do ambiente como sendo partes atrativas, repulsivas ou inertes analogamente aos campos eletromagnéticos e/ou campos gravitacionais.

Os campos potenciais são utilizados para agregar inteligência ao controle baseado em comportamento, Arkin (ARKIN, 1998) [1], Brooks (BROOKS, 1999) [48], Murphy (MURPHY, 2000) [26] e Stone (STONE, 2007) [46] contribuíram para este progresso pois aplicaram os campos potencias em diversos projetos comportamentais.

"Transformar um robô de um computador sobre rodas, meramente capaz de perceber algumas propriedades físicas do ambiente através de seus sensores, em um agente inteligente, capaz de identificar atributos, detectar padrões e regularidades, aprender a partir de experiência, localizar-se, construir mapas e navegar, necessita da aplicação simultânea de muitas disciplinas de pesquisa." (NEHMZOW, 2000) [34]

1.3. Objetivo e Contribuição

A presente dissertação tem como objetivo simular e implementar o controle baseado em comportamento de um robô móvel autônomo. Este robô terá sua percepção do ambiente gerada a partir do sensoriamento embarcado e utilizará a

arquitetura de controle de esquemas motores "*Motor-Schema*" desenvolvida por Ronald C. Arkin (ARKIN 1998) [1] que utiliza uma variação dos métodos de campos potenciais para determinar as ações a serem tomadas.

As contribuições geradas neste trabalho contemplam uma grande revisão bibliográfica sobre o assunto e elaboração de uma documentação técnica completa, pouco difundida em língua portuguesa. A bibliográfica de referencia sobre o assunto, em língua inglesa, cita a utilização de cinco comportamentos primitivos que agregados, geram três comportamentos complexos. Neste trabalho utilizaram-se dez comportamentos primitivos que combinados geram cinco comportamentos complexos, além de fornecer todos os códigos fonte utilizados nas simulações e experimentos.

1.4. Organização da Dissertação

Feita a introdução histórica e com motivação e objetivo estabelecidos, organizou-se o trabalho da seguinte forma:

Capítulo 2: Apresenta o embasamento teórico para entendimento desta dissertação e contempla os tipos de controles autônomos existentes, arquiteturas usadas no controle baseado em comportamento e a ferramenta de simulação usada.

Capítulo 3: Apresenta o hardware sensorial que foi utilizado, tanto na simulação quanto nos experimentos.

Capítulo 4: Detalha toda a simulação realizada sobre o controle baseado em comportamento.

Capítulo 5: Demonstra o experimento realizado e o compara com as simulações.

Capítulo 6: Conclusões finais da dissertação.

Capítulo 7: Referências bibliográficas.

2 Embasamento Teórico

Este capítulo apresenta o embasamento teórico para entendimento desta dissertação, e contempla os principais tipos de controles autônomos existentes, arquiteturas usadas no controle baseado em comportamento, e a ferramenta de simulação usada.

2.1. Robôs Móveis Autônomos

Grande parte dos robôs móveis e fixos foram idealizados para trabalhar em ambientes bem diferentes. Os dispositivos fixos tipicamente são especificados para ambientes estruturados, pré-definidos e relativamente estáticos. Já os robôs móveis pretendem atuar em ambientes altamente dinâmicos e desconhecidos.

Ambientes estruturados possuem objetos em posições definidas antes que qualquer processo seja iniciado, objetos irrelevantes para as tarefas são tipicamente excluídos destes ambientes pela possível dificuldade de modelá-los matematicamente de uma forma simplificada. O chão de fábrica de indústrias automobilísticas é altamente estruturado. Salas de escritórios comerciais e residências têm uma definição intermediária da natureza dos objetos incluídos no ambiente, enquanto que florestas e desertos não mapeados, por exemplo, são totalmente aleatórios em termos de pré-definição do ambiente.

Um ambiente dinâmico é um ambiente que sofre alterações constantemente. Os objetos que fazem parte do local podem alterar sua posição e orientação independentemente do robô. Robôs fixos aceitam trabalhar com ambiente com pouca dinâmica como, por exemplo, objetos que variam sua posição e orientação em uma esteira industrial, contanto que o robô seja capaz de identificar essas pequenas variações. Robôs móveis projetados para ambientes dinâmicos não possuem *a priori* o conhecimento da localização e orientação dos objetos e sua variação durante o processo.

Outra grande diferença entre ambos é a auto-localização: robôs fixos sabem exatamente onde estão devido a não sofrerem alteração de sua base instalada e

possuírem *encoders* de posição precisos (Figura 8) em suas juntas, enquanto os móveis raramente sabem, pois mesmo que possuam *encoders*, sofrem com as variações do ambiente e do solo que geram erros significativos (Figura 8).

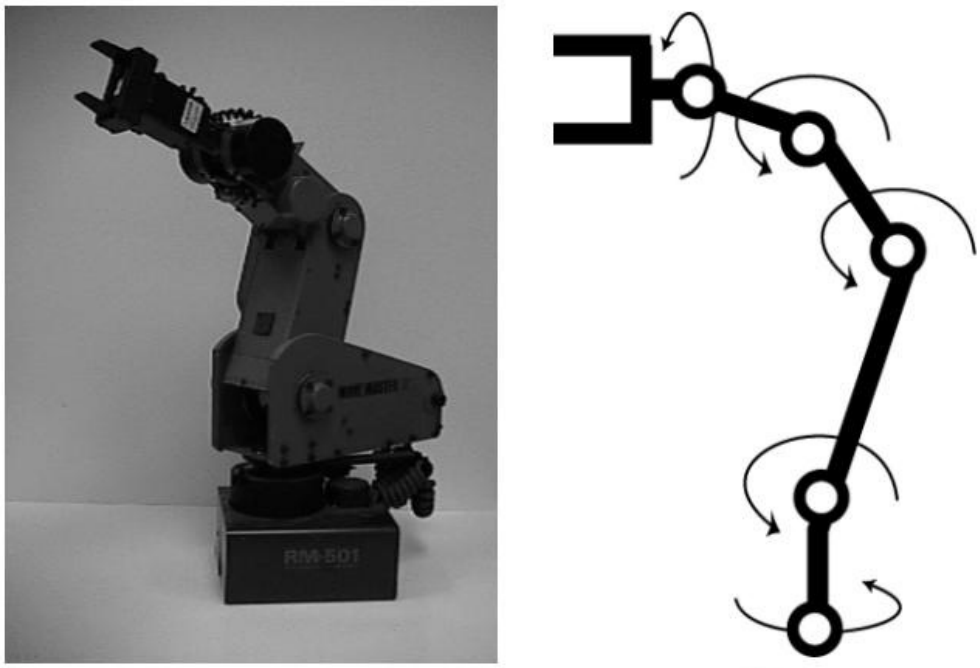


Figura 8 - Robô *MOVEMASTER* a) Robô Fixo b) Juntas associadas (MURPHY, 2000) [26]

O grande desafio que os robôs móveis autônomos trouxeram para a atualidade é dotar estes sistemas de uma capacidade de raciocínio inteligente e de interação com o meio em que estão inseridos, e fazê-los "sentir" e "reagir" ao ambiente em que estão inseridos através da leitura de seus sensores (e.g. sensores infravermelho, lasers, *bumpers*, câmeras de vídeo, etc.). É através desta percepção sensorial que podem executar melhor as suas ações, (MEDEIROS 1998) [24], (HEINEN 1999) [25].

Existem robôs móveis atuando em diferentes áreas, como por exemplo: robôs desarmadores de bombas, robôs usados para a exploração de ambientes hostis, e a condução de veículos (carros) robotizados. (Figura 9).

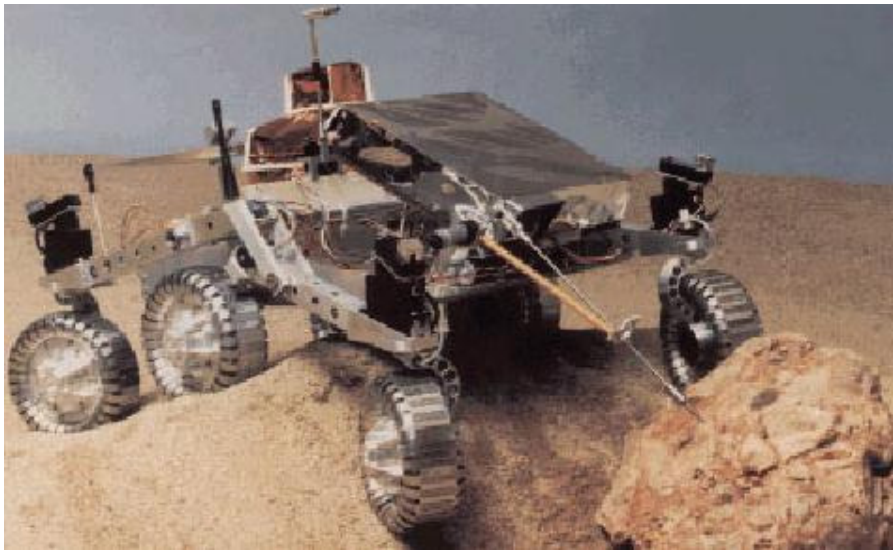


Figura 9 - Robô *Soujorner* - Robô para exploração de ambientes dinâmicos e desconhecidos (MURPHY, 2000) [26].

Exemplos de robôs móveis autônomos que obtiveram destaque internacional são: o sistema desenvolvido pelo NavLab da CMU (POMERLEAU, 1998) [27], (BATAVIA, 1996) [29] capaz de conduzir uma caminhonete pelas estradas americanas; o robô do tipo exploratório enviado para Marte pela NASA (SCHENKER, 2003) [14], o robô Dante que explora vulcões (LEMONICK ,1994) [28] e o sistema de controle de um veículo *Ligier* elétrico desenvolvido pelos pesquisadores do INRIA na França (PAROMTCHIK, 1996) [30], (SCHEUER, 1998) [31]. Os robôs exploratórios enviados para Marte chamam atenção pela alta tecnologia autônoma empregada, em 2003 a NASA enviou os robôs *Spirit* e *Opportunity* (NASA, 2003) [56] e o robô *Phoenix* (NASA, 2007) [56]. Aplicações militares como o BIGDOG (BOSTON D., 2008) [57], robô de carga para terrenos acidentados. Destaque também para o entretenimento pessoal como o robô AIBO da empresa *SONY* (STONE, 2007) [46] que simula um cachorro e suas emoções, e para robôs de limpeza doméstica como o *Roomba 500* da empresa *iRobot* (iROBOT, 2007) [55]. Todos estes sistemas possuem em comum a capacidade de receber leituras de sensores que lhes dão informações sobre o ambiente em que estão inseridos e de modo parcialmente ou completamente autônomo. Geram os comandos de deslocamento para navegação em um ambiente de modo seguro, sem se chocar contra obstáculos ou colocar em risco sua integridade ou a dos elementos no ambiente.

2.2. Sistemas de Controle

Para executar uma tarefa, um robô normalmente deve agregar tipos diferentes de objetivos. Buscar um determinado objeto num ambiente desconhecido é uma tarefa de alto nível que se utiliza de vários objetivos enquanto que uma tarefa do tipo "andar em linha reta com velocidade constante" é denominada uma tarefa de baixo nível, pois utiliza um ou poucos objetivos agregados.

Precisa-se estabelecer quais são os elementos que interagem com o sistema de controle para poder defini-lo. Ligado diretamente ao sistema de controle está o robô, que é o sistema que deseja-se controlar. O ambiente não é ou não pode ser totalmente controlado diretamente pelo sistema de controle, mas interfere de maneira significativa em seu funcionamento.

O sistema de controle tem a tarefa de fazer com que todo o sistema alcance um determinado estado. Alcançar este estado pode envolver ou depender de mudanças que ocorrem no ambiente, no sistema controlado ou devido à interação entre os dois. O sistema de controle trabalha adquirindo informações captadas do ambiente através de sensores, e a partir destas leituras toma decisões, atua e interage com o ambiente através de motores, garras, manipuladores, dentre outros. Logo, um sistema de controle é um processo que pode utilizar seus sensores para obter informações sobre o sistema controlado e sobre o ambiente. Ele pode utilizar este conhecimento para controlar seus atuadores fazendo com que todo o sistema alcance um determinado estado.

A fim de exemplificar a função do sistema de controle, considere o seguinte exemplo. Um robô se locomove dentro de uma arena cercada tentando evitar a colisão com muros. Neste caso, o sistema controlado é o robô, e os muros são parte do ambiente. A interação do robô com o ambiente se dá com o contato das rodas com o terreno ou caso o robô acidentalmente colida com um muro. O sistema de controle obtém informações sobre o próprio robô e sobre o ambiente através dos sensores, as processa, e então utiliza a resposta gerada por este processamento para controlar os atuadores e manter o robô distante dos muros, que seria o estado desejado do sistema.

2.3. Técnicas de Controle

As técnicas de controle mais difundidas e utilizadas por um sistema de controle são denominadas controle em malha aberta (*open loo*p) e malha fechada (*closed loop*). Os sistemas *open loop* não necessitam de sensores. Admite-se por exemplo um sistema de controle que deve fazer um robô se mover a uma velocidade constante. Após os cálculos preliminares com base nos dados físicos do robô, pode-se utilizar um modelo que pode prever quanta energia deve ser fornecida aos motores do robô para que ele atinja a velocidade desejada. Esta técnica não garante que o robô vá manter sua velocidade constante, pois ao se iniciar o processo não há mais como alterar seus parâmetros caso haja alguma modificação no ambiente.

Os sistemas *closed loop* podem ser divididos em sistemas de malha fechada *feedforward* ou *feedback*. Os sistemas que utilizam o f*eedforward* utilizam sensores somente para perceber o ambiente. Neste tipo de técnica de controle, medições do ambiente são utilizadas para atualizar variáveis no modelo do sistema. A partir do exemplo anterior, pode-se adicionar um sensor que informa a inclinação do terreno, pois esta informação pode ser utilizada para recalcular a força necessária para que o robô atinja a velocidade desejada.

As técnicas *open loop* e malha fechada *feedforward* podem ser utilizadas preferencialmente quando o ambiente é praticamente estático e previsível, o que não ocorre na maioria dos casos de controle robótico. Ambientes dinâmicos necessitam de um controle mais fino, e a técnica mais utilizada em sistemas robóticos é de malha fechada *feedback*. Esta monitora continuamente a resposta dos sensores e ajusta seus atuadores de acordo com a necessidade programada. O exemplo citado pode ser alterado substituindo o sensor de inclinação por um *encoder* que irá determinar a velocidade atual das rodas do veículo e com esta informação atuar para que, assumindo que não haja deslizamento, o mesmo mantenha sempre a mesma velocidade, independentemente de inclinações na pista que possam variar.

2.4. Arquiteturas de Controle

A definição de arquitetura de controle robótico segundo Russel (RUSSELL 2003) [32] é:

"A arquitetura de um robô define como é organizada a tarefa de gerar ações através da percepção."

As principais arquiteturas são descritas a seguir.

2.4.1. Arquitetura Horizontal

Tradicionalmente, para o desenvolvimento de sistemas de controle para robôs móveis autônomos, utiliza-se a arquitetura horizontal, que tem como premissa dividir o controle em unidades de função. Cada uma destas unidades está intimamente ligada e dependente às unidades vizinhas, de tal forma que todo o sistema deve ser projetado de uma forma completa, pois um nível sozinho não é capaz de executar nenhuma ação. Para se adicionar uma nova atividade, é preciso alterar todo o projeto em sua concepção.

Neste tipo de arquitetura, as tarefas do sistema de controle são divididas em várias tarefas mais simples baseadas em suas funcionalidades. Uma abordagem comum é dividir as tarefas como mostra a figura 10.

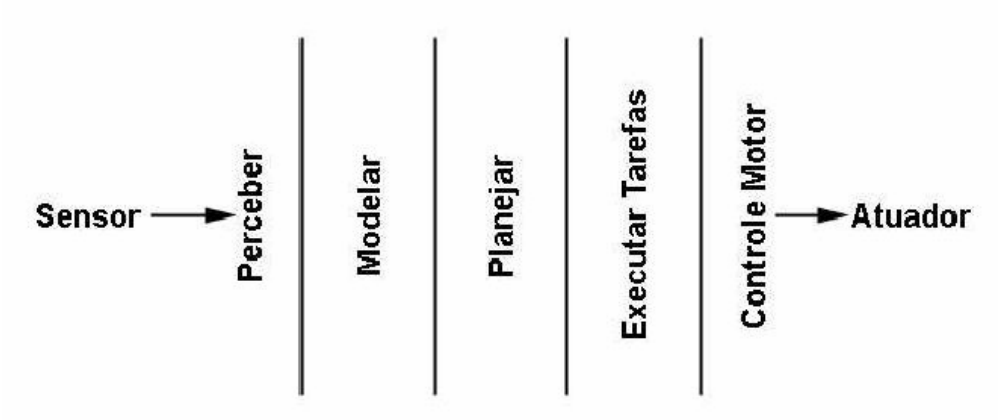


Figura 10 – Arquitetura Horizontal de Controle. Modelo SMPA *(Sense, Model, Plan, Act)*

Os sistemas de controle baseados nesta arquitetura executam suas tarefas em várias etapas. As entradas sensoriais são utilizadas para modificar a representação interna do ambiente e logo após e baseado nesta representação, um plano a longo prazo é elaborado. Esta sequência resulta em uma série de ações que o robô deve executar para alcançar o seu objetivo. Esta série de ações é utilizada para comandar os atuadores do robô e encerrar o ciclo de controle. A partir deste ponto o sistema é reiniciado para atingir novos objetivos.

Obter as informações relevantes do ambiente e construir um modelo o mais completo possível é a parte fundamental desta arquitetura. Feito este modelo estático do mundo real, os algoritmos planejam da maneira mais eficiente que podem as ações necessárias para alcançar os objetivos. Problemas diversos surgem com esta abordagem. Armazenar um modelo, na maioria dos casos, é complexo, devido às limitações e imperfeições dos sensores. Um ambiente dinâmico inviabiliza o planejamento gerado, assim como o fato de que enquanto o sistema de controle está planejando a próxima seqüência de ações, ele não é capaz de perceber as mudanças no ambiente. Caso algo diferente ocorra no ambiente enquanto o sistema está planejando, estas diferenças não serão consideradas no plano, o que resulta, no mínimo, em um plano defasado em relação à realidade atual, mas possivelmente em um plano perigoso para o robô e para o ambiente em questão.

2.4.2. Arquitetura Vertical

A arquitetura vertical, utilizada neste trabalho, foi idealizada pela primeira vez por Rodney Brooks em 1986 (BROOKS 1986) [16], que em sua definição, denominou de arquitetura "*subsumption*". Será utilizada a arquitetura "*subsumption*" como exemplo de arquitetura vertical, e as descrições serão baseadas em uma arquitetura vertical mais genérica. O termo tem tradução literária para subsunção, é sinônimo de supressão ou ato de suprimir.

O desenvolvimento desta arquitetura difere notavelmente da maneira tradicional de dividir o controle em módulos de função, como percepção, planejamento, modelagem, etc.

Brooks propôs que, ao invés de as tarefas serem divididas em função da funcionalidade, a divisão deveria ser feita baseando-se em comportamentos que executam tarefas, organizados em camadas. A organização do controle se dá na criação de um conjunto de camadas que representam uma tarefa ou comportamento complexo. (Figura 11).

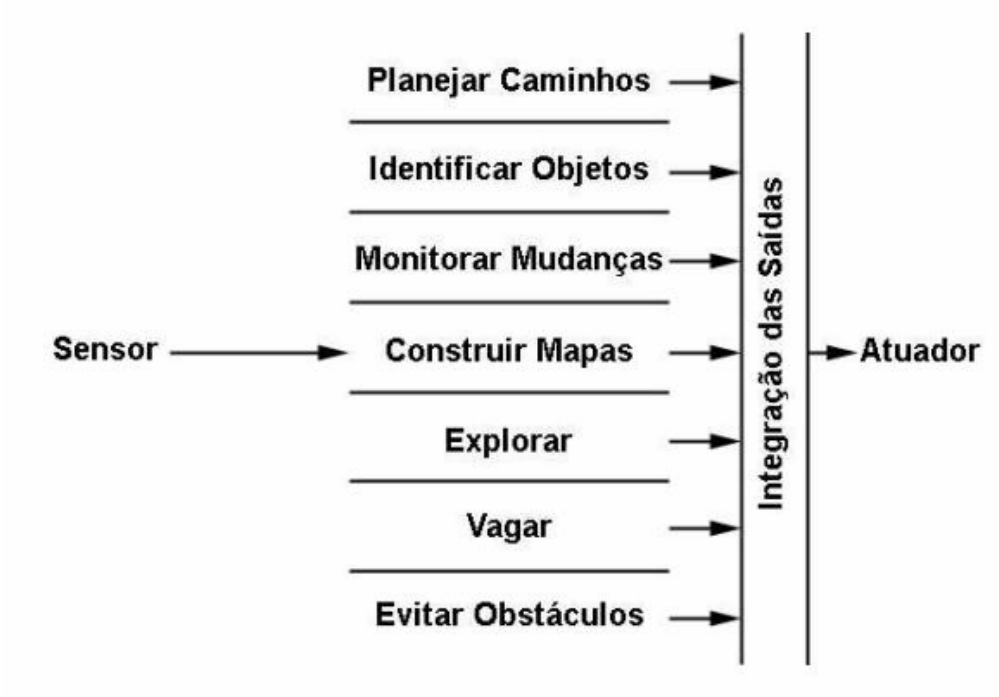


Figura 11 – Arquitetura Vertical de Controle.

A divisão em camadas de atividades permite acrescentar um comportamento quando este for necessário, construindo a nova camada e relacionando-a com o sistema. A idéia é construir um sistema autônomo completo, muito simples, e testá-lo no mundo real. (Figura 12).

Figura 12 – Arquitetura *Subsumption*.

Subsumption, em inglês, vem de subsunção ou supressão, e encaixa perfeitamente neste contexto, pois a camada mais elevada suprime a função das camadas inferiores quando uma determinada configuração dos sensores indica uma situação favorável para sua atuação. O nível acima suprime o fluxo de dados da camada inferior. É possível interromper o sistema em qualquer camada, pois as

camadas inferiores continuarão formando um sistema completo e funcional, porém com menos “inteligência”.

Um sistema de controle que utiliza a arquitetura *subsumption* é constituído de diversos comportamentos executados em paralelo, e cada um destes apresenta uma resposta de orientação diretamente para os atuadores utilizando as entradas sensoriais. As saídas sugeridas pelo comportamento com a mais alta prioridade suprimem as saídas de baixa prioridade, e são então utilizadas para controlar os atuadores do robô. Não existe relação combinatória entre as saídas de camadas diferentes. As saídas são sequenciadas conforme a prioridade da hierarquia.

A principal novidade desta arquitetura foi que uma entrada sensorial não precisa mais passar por uma série de camadas de processamento antes de se transformar em uma saída para os atuadores. Somente as entradas sensoriais relevantes para a tarefa executada por aquele determinado comportamento são utilizadas, não para atualizar uma representação interna, mas para gerar diretamente as saídas para os atuadores. Isto torna a ligação entre os sensores e os atuadores mais forte e direta que resulta em alta velocidade de resposta.

A técnica "*open loop*" utilizada na arquitetura horizontal é abandonada e os sistemas baseados na arquitetura vertical são na maioria construídos utilizando-se uma técnica de controle do tipo "*feedback*".

Atividades simples dos níveis inferiores provocam reações rápidas devido à não utilização de representações complexas do mundo por parte das camadas. O sistema está baseado no principio de percepção-ação. A idéia é sentir o ambiente frequentemente e assim ter uma idéia atualizada e em tempo real do que acontece.

Cada camada pode ser imaginada como tendo seu propósito implícito. A idéia principal é adaptar cada objetivo às condições atuais do ambiente real.

2.4.3. Arquitetura Híbrida

A integração dos componentes da arquitetura horizontal para o planejamento a longo prazo, e resolução de problemas com os componentes da arquitetura vertical para o controle em tempo real, é denominada arquitetura híbrida.

Esta arquitetura pode ser classificada entre a vertical e a horizontal por combinar aspectos de ambas. A abordagem híbrida não traz nenhuma estratégia

isolada adequada para a execução de todas as tarefas relevantes para um sistema de controle robótico, o que traria complicações na elaboração do planejamento e execuções em tempo real das tarefas. Apenas problemas muito específicos teriam necessidade do uso deste tipo de arquitetura.

2.5. Estratégias de Controle

"A estratégia de controle fornece os princípios para a organização dos sistemas de controle. Além de fornecer a estrutura, que impõe restrições sobre a forma como problema de controle pode ser resolvido." (MATARIC 1997) [33]

Mataric também resumiu a maneira como as estratégias mais utilizadas se comportam:

Controle Deliberativo – "Pense, depois aja"
Controle Reativo – "Não pense, reaja"
Controle Híbrido – "Pense e atue ao mesmo tempo"
Controle Baseado em Comportamento – "Pense na maneira de atuar"

As estratégias de controle são descritas a seguir.

2.5.1. Controle Deliberativo

Pensar e depois agir resume a idéia do controle deliberativo, pois ele contém explicitamente representado um modelo simbólico do mundo, e as decisões são planejadas previamente para só após serem colocadas em prática.

O robô usa todas as informações sensoriais e todo o conhecimento armazenado internamente para construir um modelo mais completo possível do ambiente, eventualmente usando fusão de sensores. A partir deste modelo, o robô gera um plano para conseguir atingir seus objetivos, e finalmente executa este plano.

O sistema de controle normalmente é organizado por uma decomposição em módulos do processo de decisão, composto por um módulo de processamento sensorial, um para modelagem, um de planejamento, um módulo de tomada de decisões, e um módulo de execução. Esta fragmentação funcional permite que

operações complexas sejam executadas, mas implica em forte interdependência sequencial entre os módulos de tomada de decisão.

Embora essa técnica tenha demonstrado alta capacidade de complexidade, suas limitações logo se tornaram evidentes. O módulo de planejamento, um dos principais componentes da inteligência artificial, é um processo que exige muito poder computacional se comparado aos outros módulos. Requer que o robô construa uma sequência de sentir, modelar, planejar e agir, ou seja, combinar os dados sensoriais em um mapa do mundo, planejar uma trajetória ótima e então enviar as saídas geradas para as rodas do robô. Brooks (BROOKS 1986) [16] se refere a esta estratégia de controle como o modelo SMPA (*Sense - Model - Plan - Act*). Este módulo deve avaliar potencialmente todos os planos possíveis até encontrar um que lhe permita atingir seu objetivo, resolver a tarefa, ou decidir sobre a trajetória a ser executada.

O controle deliberativo tem dificuldade de operar em um ambiente dinâmico, pois ruídos ou imprecisões dos sensores causam grandes erros acumulados e o modelo de mundo torna-se ineficaz, o que gera a necessidade de uma estratégia de controle diferenciada.

2.5.2. Controle Reativo

O controle reativo é fundamentado na biologia, e utiliza a concepção do estímulo e resposta, que não exige o conhecimento prévio do ambiente a ser explorado e não depende dos processos complexos utilizados no controle deliberativo. É um método de controle poderoso e eficaz que se inspira na natureza. Insetos, que superam amplamente os vertebrados em número, são altamente reativos.

Esta estratégia utiliza como base a arquitetura vertical para dividir o controle em uma série de regras que concorrem entre si. Estas regras envolvem uma quantidade mínima de computação, e não utilizam representações internas ou conhecimento global do mundo.

Sistemas reativos podem alcançar rápidas respostas em tempo real pela incorporação no controlador do robô de uma coleção de regras pré-programadas, com estados simples definidos como, por exemplo, uma regra que define que em caso de qualquer colisão os motores devem ser parados, e outra regra que em caso

de parada dos motores deve-se recuar. Cada um destes processos é conectado com suas próprias entradas sensoriais, com a possibilidade de inibir as entradas ou saídas de outros comportamentos, dependendo das prioridades de cada um. Isso faz o controle reativo especialmente adequado para mundos dinâmicos e não-estruturados, em que ter acesso a um modelo do mundo não é uma opção viável. Além disso, a quantidade mínima de computação embutida nos sistemas reativos torna a resposta rápida às mudanças no ambiente.

Braitenberg (BRAITENBERG, 1984) [43], utilizando o ponto de vista da psicologia, estendeu os princípios do comportamento de circuitos analógicos para uma série de veículos reativos. Estes sistemas utilizam inibidores e excitadores diretamente acoplados entre os sensores e os motores. Braitenberg criou veículos que simulam de forma primitiva reações animais como medo, agressividade (Figura 13) e até mesmo amor. (Figura 14).

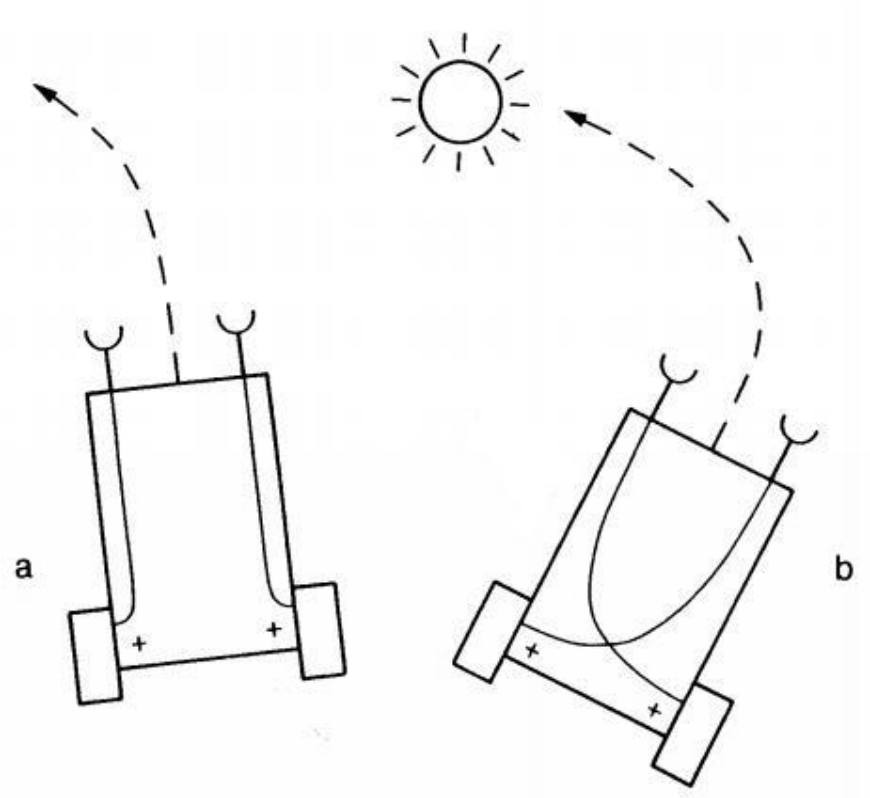


Figura 13 – Robôs reativos. a) Expressa medo. b) Expressa agressividade.

Os robôs apresentados na figura 13 possuem dois motores e dois sensores foto-elétricos ligados diretamente aos motores, ou seja, quanto maior a luminosidade, maior é a tensão aplicada ao respectivo motor. A figura 13(a)

expressa um comportamento de medo da luz devido à proximidade com a fonte de luz, e o robô se locomove para longe dela. Já na figura 13(b), os sensores são ligados inversamente aos motores, o que traduz um comportamento de agressão, pois quanto mais próximo da luz, mais rápido o robô irá se aproximar até colidir.

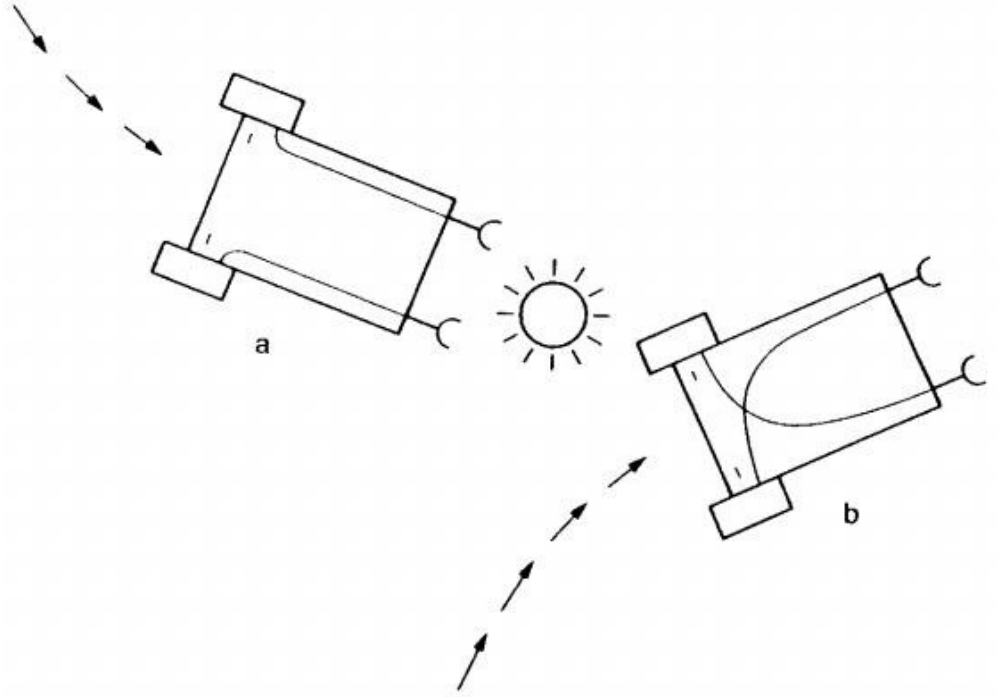


Figura 14 – Robôs reativos. a) Expressa amor. b) Expressa curiosidade.

Na figura 14, o mesmo esquema de ligação é utilizado, porém inibidores são usados entre os sensores e os motores, ou seja, quanto maior a luminosidade, menor é a velocidade transmitida para as rodas, e a ausência de luminosidade faz com que o robô pare. Este fato traduz na figura 14(a) um comportamento denominado de "amor" pela fonte luminosa, pois o robô ao detectar a fonte luminosa se aproxima até parar, e ficará parado indefinidamente, como se contemplasse a fonte. Já na figura 14(b), com as ligações invertidas o robô desvia lentamente ao se aproximar da fonte luminosa até não detectá-la mais, o que demonstra curiosidade por estar sempre próximo à fonte luminosa, mas sempre virado para outro lado aguardando alguma outra fonte, um comportamento denominado de "curiosidade" ou de "exploração".

Em geral, um sistema reativo não possui nenhuma forma de representação interna do modelo do ambiente, e não utiliza um raciocínio simbólico complexo. Ele é baseado no princípio da reatividade, ou seja, na suposição de que

comportamentos inteligentes podem ser gerados sem nenhuma representação simbólica explícita e de que a inteligência é uma propriedade que emerge de certos sistemas complexos.

No entanto, as limitações à reatividade pura incluem a incapacidade de armazenar as representações internas do mundo e, portanto, não pode otimizar sua trajetória. O controle reativo abdica da complexidade de raciocínio para valorizar a velocidade de resposta. Em outros tipos de ambientes e tarefas, onde os modelos internos, memória ou aprendizagem são necessários, o controle reativo não é suficiente.

2.5.3. Controle Híbrido - Reativo/Deliberativo

O controle híbrido tem por objetivo combinar os melhores aspectos dos controles reativo e deliberativo. A resposta em tempo real da reatividade e a racionalidade e otimização da deliberação trazem resultados que contêm os dois componentes combinados, o reativo, condição simultânea de regras de ação, e o deliberativo, que deve interagir para produzir uma saída coerente.

Combinar estas estratégias é um desafio porque o componente reativo lida com as necessidades imediatas do robô, tais como deslocamento, evitando obstáculos e, portanto, opera em uma escala de tempo muito rápida e usa diretamente dados sensoriais. Já o componente deliberativo utiliza uma representação altamente abstrata e simbólica do mundo, e funciona numa escala de tempo mais longa. Enquanto os resultados das duas componentes não estão em conflito, o sistema não exige uma coordenação maior.

As duas partes do sistema devem interagir para que possam beneficiar o sistema como um todo, e consequentemente o sistema reativo deve suprimir o deliberativo se o mundo apresentar algum desafio inesperado e imediato. Analogamente, a componente deliberativa deve fornecer informações à reativa, a fim de guiar o robô em direção a trajetórias mais eficientes e melhores até seu objetivo. A interação das duas partes do sistema exige um componente intermediário, que concilia as diferentes representações utilizadas pelos outros dois e os conflitos entre as suas saídas. A construção deste componente intermediário é geralmente o maior desafio do projeto do sistema híbrido.

O controle híbrido é referenciado como sendo uma estratégia de três camadas, devido à sua estrutura, que consiste na execução, coordenação e planejamento, que são respectivamente as camadas reativa, intermediária e deliberativa.

Duas arquiteturas clássicas que utilizaram a estratégia híbrida em sua concepção foram a *AuRA* (ARKIN 1986) [1], e a arquitetura *Atlantis* (GAT 1991) [35]. Ambas utilizam o controle em três camadas.

Arquiteturas de três camadas têm o objetivo de aproveitar o melhor do controle reativo no que diz respeito à dinâmica do ambiente e velocidade de resposta, com o melhor do controle deliberativo, com ações globalmente eficientes sobre uma escala de tempo. No entanto, existem questões complexas envolvidas na integração dessas estratégias fundamentalmente diferentes, e na maneira em que a sua funcionalidade deve ser dividida.

2.5.4. Controle Baseado em Comportamento

O controle baseado em comportamento tem sua inspiração obtida da biologia, e tenta modelar o cérebro dos animais para lidar com problemas complexos, tanto de planejamento quanto de execução de ações. É constituído de um conjunto de módulos de interação, chamados de comportamentos, que coletivamente tentam atingir um nível complexo de comportamento. Para um observador externo, os comportamentos são os padrões de atividade do robô emergentes das interações entre o robô e seu ambiente. Um programador vê os comportamentos como módulos de controle que definem um conjunto de ferramentas para alcançar um objetivo.

Cada comportamento recebe entradas de sensores, de outros comportamentos do sistema, ou de ambas, e oferece saídas para os atuadores do robô ou para outros comportamentos. Assim, um controle baseado em comportamento é uma rede estruturada de interações entre comportamentos, sem uma representação centralizada do mundo ou foco de controle. Em vez disso, os comportamentos individuais e redes de comportamentos garantem quaisquer informações sobre o estado e modelos.

"O controle comportamental permite relacionar, de maneira coerente, todos os elementos que um controle robótico exige." (JONES, 1998) [20]

Sistemas baseados em comportamento bem projetados aproveitam a dinâmica de interação entre os comportamentos e entre os comportamentos e o ambiente. A funcionalidade destes sistemas emerge dessas interações e, portanto, não são propriedades do robô ou do ambiente, e sim um resultado da interação entre eles.

Ao contrário do controle reativo, que utiliza conjuntos de regras reativas com pouco ou nenhum controle do estado e sem representação, o controle baseado em comportamento utiliza conjuntos de comportamentos que não possuem essas restrições. Comportamentos têm estados e podem ser usados para construir representações, permitindo o raciocínio, planejamento e aprendizagem.

Relacionado ao controle híbrido que precisa de um modelo do ambiente e trabalha em uma escala de tempo mais lenta o controle baseado em comportamento ganha velocidade de processamento por não ter esta representação armazenada. Outro diferencial é que no controle por comportamento não existe um processo que precisa coordenar partes reativas com deliberativas, que é uma tarefa complexa e possui dependências dos módulos de função do robô, que não permite adições de outros comportamentos de maneira modular.

O intuito deste trabalho é utilizar o controle baseado em comportamento para explorar ambientes dinâmicos e desconhecidos sem um modelo pré-definido do mesmo.

O controle baseado em comportamento possui algumas características bem definidas, entre elas:

- Comportamentos são implementados como as leis de controle (por vezes semelhantes às usadas na teoria de controle), seja em software ou hardware, como um elemento de processamento ou como um processo.
- Cada comportamento pode receber entradas de sensores do robô (sensores de proximidade, detectores de ultrassom, sensores de contato, câmera) e de outros módulos. As saídas são enviadas para os atuadores do robô (rodas, garras, braços, fala) e outros módulos.
- Comportamentos diferentes podem receber contribuições de forma independente do mesmo sensor e comandar de ações de saída para o mesmo atuador.

- Os comportamentos são codificados para serem de relativa simplicidade, e são adicionados ao sistema de forma incremental.
- Comportamentos (ou seus subgrupos) são executados simultaneamente e não sequencialmente, a fim de explorar o paralelismo e velocidade de computação, bem como a dinâmica de interação entre os comportamentos e entre os comportamentos e o ambiente.

A robótica baseada em comportamento foi desenvolvida para permitir ao robô adaptar-se à dinâmica do ambiente do mundo real sem trabalhar sobre abstrações da realidade, mas também dar-lhe uma capacidade computacional maior da que está presente nos robôs reativos. Sistemas baseados em comportamento possuem uma forte ligação entre o sentir e o agir devido aos comportamentos. É raro para um comportamento realizar cálculos extensos confiando em um modelo de ambiente tradicional, a menos que tal cálculo possa ser feito em tempo hábil para uma resposta dinâmica que exija rapidez na saída.

Os comportamentos são projetados em diferentes níveis de abstração, facilitando a construção destes sistemas. Novos comportamentos são introduzidos no sistema de forma incremental e modular, desde o simples ao mais complexo, até que a interação entre eles resulte na capacidade total desejada para o robô. (Figura 15).

Comportamento para desviar de obstáculos estáticos assim como desviar especificamente de “predadores” e o comportamento que gera um ruído aleatório criado para escapar situações singulares, quando executados em paralelo gera um comportamento mais completo como o comportamento explorar. (Figura 15).

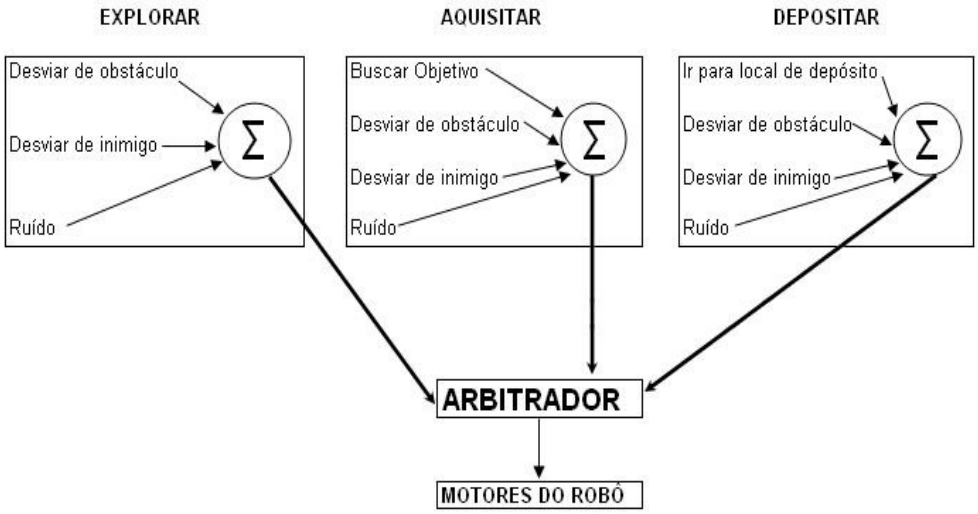


Figura 15 – Comportamentos complexos gerados a partir da interação entre comportamentos simples.

O primeiro passo é implementar os comportamentos primários de sobrevivência, como evitar obstáculos e colisões iminentes. Estes comportamentos são muitas vezes de natureza reativa, uma vez que as regras reativas são componentes de comportamentos simples. Nota-se que existe uma distinção entre as condições de ativação, que permitem ao comportamento gerar estímulos, a partir dos quais as ações são geradas.

Posteriormente, novos comportamentos são adicionados para fornecer capacidades mais complexas, como seguir paredes, perseguir um alvo, encontrar um objeto, recarregar uma bateria, evitar luz, juntar-se a um grupo, pegar um objeto, encontrar um marco definido, fugir de um "predador".

A interação e integração dos efeitos temporais e espaciais são de fundamental importância no comportamento destes sistemas. O efeito combinado dos comportamentos executados paralelamente ao longo do tempo, orientados pela percepção e estados internos, cria o comportamento relevante à dinâmica de um sistema de controle baseado em comportamento.

Os sistemas baseados em comportamento devem resolver o problema da escolha de uma determinada ação ou comportamento a partir de várias opções. Este é um dos desafios centrais da concepção baseada em comportamento. Uma abordagem para seleção de ação é o uso de uma hierarquia de comportamento pré-definido, em que os comandos do mais alto nível ao comportamento ativo são enviados para o atuador e todos os outros de nível inferior são ignorados. Existem

várias outras metodologias baseadas em outros princípios para solucionar o problema da seleção de ação, como a arquitetura de esquemas motores. Estas metodologias têm como objetivo proporcionar uma maior flexibilidade mas, em alguns casos, podem fazê-lo com o custo de reduzir a eficiência ou a capacidade de análise do sistema de controle resultante.

2.5.4.1. *Motor-Schema* e Campos Potenciais

Dentro do controle baseado em comportamento existem diversas arquiteturas possíveis para organização dos comportamentos, dentre elas destacam-se:

- "S*ubsumption*" por Rodney Brooks em 1986 [16]
- "*Motor-Schema*" por Ronald Arkin em 1987 [38]
- "*Action-selection*" por Pattie Maes em 1990 [49]
- "*Colony*" por John Connell em 1990 [50]
- "*Animated agent*" por R.James Firby em 1989 [54]
- "*DAMN (Distributed Architecture for Mobile Navigation)*" por Julio Rosenblatt em 1997 [51]
- "*Skill Network*" por David Zeltzer em 1991 [53]
- "*Circuit*" por L. Kaelbing e S. Rosenschein em 1991 [52]

Cada método possui características que se adequam melhor a diferentes tipos de finalidades para as quais o robô será programado. A arquitetura usada neste trabalho é a de esquemas motores (*Motor-Schema*) desenvolvida por Ronald Arkin (ARKIN 1987) [36], a mais difundida no controle baseado por comportamento.

A teoria dos esquemas motores fornece os seguintes recursos para a especificação e concepção de sistemas baseados em comportamento (ARBIB 1992) [37]:

- Explica o comportamento em termos de controle simultâneo de muitas atividades diferentes.
- Um esquema armazena como reagir e a forma que a reação deve ser realizada.

- É um modelo distribuído de computação.
- Fornece uma linguagem para conectar ação e percepção.
- Os níveis de ativação são associados com os esquemas que determinam a sua disponibilidade ou a aplicabilidade de atuação.
- É útil para explicar o funcionamento do cérebro, bem como aplicações distribuídas de inteligência artificial (como a robótica baseada em comportamento).

A teoria dos esquemas motores na robótica móvel autônoma, segundo Arkin (ARKIN 1987) [36], fornece grande modularidade computacional ao sistema para expressar a relação entre controle motor e percepção, em contraste com os modelos de redes neurais. Os esquemas motores atuam individualmente, porém todos em paralelo, ou seja, são agentes distribuídos em uma competição cooperativa que os torna facilmente mapeáveis em arquiteturas distribuídas. Comportamentos primitivos gerados pelos esquemas motores, quando agrupados, podem gerar comportamentos mais complexos.

O método de esquemas motores aplicado na robótica móvel fornece comportamentos reativos que, ao trabalharem de forma simultânea, produzem resultados inteligentes e comportamentos complexos em resposta a estímulos ambientais.

As diferenças e principais vantagens desta arquitetura para as demais existentes são:

- As saídas geradas pelos comportamentos são representadas em um modelo único de vetores contendo velocidade e ângulo do robô, gerados a partir de uma abordagem de campos potenciais, também discutida neste trabalho.
- A adição dos vetores, ou seja, a soma de todas saídas, gera a melhor coordenação para o destino do robô.
- Não existe hierarquia pré-definida para a coordenação, em vez disso os comportamentos são configurados em tempo real a partir das interações do robô com o ambiente. Os esquemas podem ser instanciados ou não a qualquer momento, com base nos eventos perceptivos; essa estrutura se parece mais com uma rede dinâmica do que uma arquitetura em camadas.

- Arbitrar simplesmente está fora do escopo desta arquitetura, ao invés disso cada comportamento pode contribuir em graus variados para a resposta global do robô através dos ganhos relativos de cada comportamento.
- As incertezas perceptivas podem ser refletidas na resposta do comportamento, permitindo que estas sejam usadas como entrada para o cálculo comportamental.

Esta teoria teve como base de desenvolvimento os componentes reativos do projeto *AuRA* (ARKIN 1986) [1] e tem seu principal método de desenvolvimento baseado na etologia , ou seja, o estudo do comportamento animal.

Esquemas motores têm grande usabilidade em várias circunstâncias. Muitos dos comportamentos têm parâmetros internos que proporcionam maior flexibilidade na sua implantação. Os comportamentos geralmente são análogos aos comportamentos animais, pelo menos aqueles úteis para tarefas de navegação.

Um esquema de percepção está inserido dentro de cada esquema motor. Esses esquemas perceptivos fornecem as informações ambientais específicas para um comportamento específico. O princípio básico do esquema perceptivo contempla o algoritmo que fornece as informações necessárias para um determinado comportamento responder conforme desejado.

Cada esquema motor possui um processo perceptivo capaz de oferecer estímulos adequados o mais rapidamente possível. Esquemas perceptivos são definidos de forma recursiva, isto é, sub-esquemas perceptivos podem extrair pedaços de informações, que são posteriormente processados por um esquema perceptivo em uma unidade comportamental mais significativa. Um exemplo desta aplicação é o reconhecimento de uma pessoa com mais de um sensor. Sensores infravermelhos podem fornecer uma assinatura de calor enquanto que a visão computacional pode fornecer uma forma humana.

As informações geradas em cada um desses processos de nível inferior da percepção são fundidas em uma interpretação de nível mais alto antes de atuar nos motores do robô. Isso permite o uso de múltiplos sensores dentro do contexto de um único comportamento.

Cada esquema motor tem como saída um vetor de ação (que neste trabalho consiste em componentes de orientação e magnitude da velocidade) que define a

forma como o robô deve se mover em resposta aos estímulos recebidos. (Figura 16).

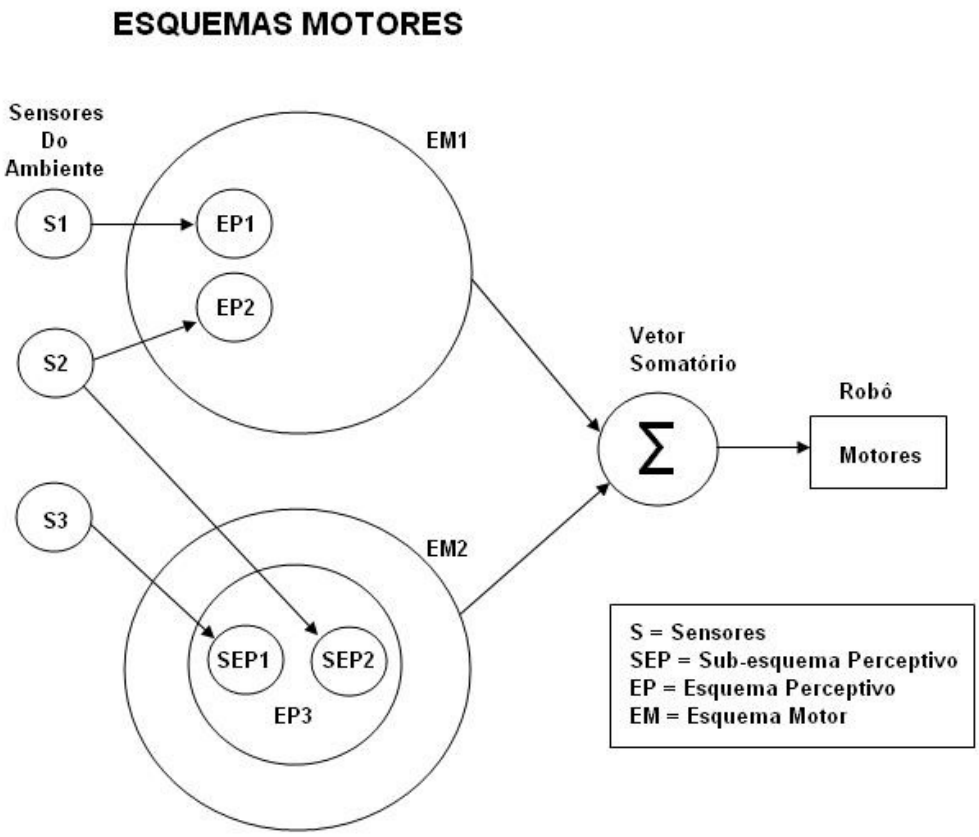


Figura 16 – Diagrama representativo da arquitetura *Motor Schema.*

Esta abordagem tem sido utilizada para navegação em terrenos planos, onde cada vetor é bidimensional (ARKIN 1989) [38], para a geração de três vetores tridimensionais para uso em voo ou navegação submarina (ARKIN, 1992) [39], e para uso em manipuladores móveis com muitos graus de liberdade redundantes (CAMERON 1993) [40].

Existem diversos esquemas clássicos já definidos, por exemplo:

- *Move-ahead*: mover-se em uma direção determinada.
- *Move-to-goal*: em direção a um objeto detectado.
- *Avoid-static-obstacle*: se afastar de objetos estáticos.
- *Dodge*: desviar de um objeto balístico que não altera trajetória.
- *Escape*: desviar de um ponto pré-determinado ou de um possível encontro com um predador que esteja continuamente em perseguição.

- *Stay-on-path*: manter-se no sentido do caminho como estradas e corredores. Para navegação tridimensional torna-se um esquema de *stay-in-channel*.
- *Noise*: move-se em uma direção aleatória.
- *Follow-the-leader*: move-se para um ponto determinado próximo a algum objeto em movimento. Comporta-se como se estivesse vinculado ao objeto.
- *Probe*: move-se para áreas abertas.
- *Dock*: abordagem de um objeto a partir de uma determinada direção.
- *Avoid-past*: afastar-se das áreas visitadas recentemente.
- *Move-up, move-down, maintain-level*: mover-se para cima, para baixo ou manter altitude.
- *Teleautonomy*: permite que o operador humano interaja e forneça dados ao sistema de controle no mesmo nível que outro esquema motor.

O método de coordenação utilizado é o do somatório e normalização dos vetores de saída, e o método de codificação da resposta dos esquemas motores é o uso contínuo de campos potenciais.

Uma partícula carregada atravessando um campo magnético ou uma bola rolando de uma colina são visualizações de campos potenciais reais. A idéia básica dos campos potenciais, apresentada por Khatib (KHATIB, 1985) [41] e Krogh (KROGH, 1984) [42], é a utilização de funções potenciais para criação de campos de forças que representam a direção e angulação que o robô deve navegar, representados por vetores com as componentes de magnitude da velocidade e direção. Objetivos são associados a campos de atração enquanto que obstáculos a campos de repulsão. O comportamento exibido pelo robô dependerá da combinação e da forma dos campos.

Uma característica marcante deste método é a codificação da ação de forma contínua, ou seja, em todo o plano haverá vetores de força associados ao objeto, não importando tamanho ou forma.

Os campos potenciais mais utilizados no controle baseado em comportamento que utiliza a arquitetura de esquemas motores são o uniforme, o

perpendicular, o atrativo, o repulsivo e o tangencial como mostrado na figura 17: (MURPHY 2000) [26]

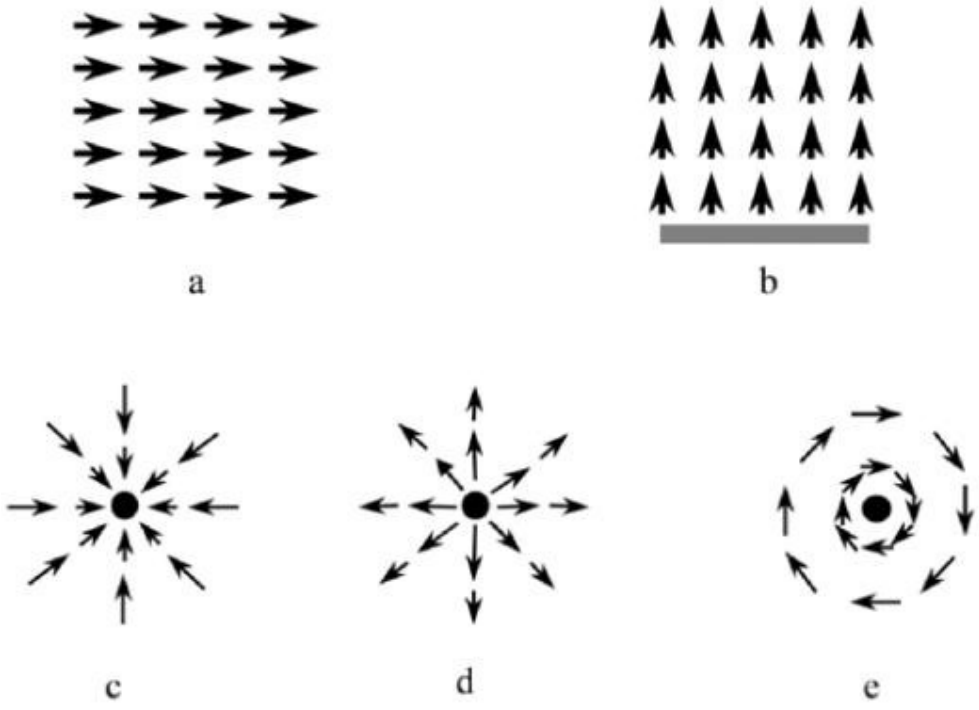


Figura 17 – Campos potenciais mais comuns. a) Uniforme b) Perpendicular c) Atrativo d) Repulsivo e) Tangencial

- Campo uniforme – Um robô em um campo uniforme sofre as mesmas forças em magnitude e direção em qualquer lugar que ele esteja. Qualquer movimento que for realizado pelo robô será guiado pelo campo, e acompanhará a direção do campo com a velocidade constante associada a ele. Este tipo de campo é usado quando se deseja que o robô tenha forças que o conduzam sempre em uma determinada direção.
- Campo perpendicular – Neste campo o robô é orientado a se afastar de um determinado obstáculo ou parede. Este campo pode estar associado a uma função decrescente, fazendo com o que o robô sofra uma força de repulsão que decresce conforme se afasta da parede.
- Campo atrativo – Um objeto de destino gera um campo radial de atração para o robô. Onde quer que o robô esteja, ele sofrerá uma força de atração na direção do objeto em questão. São campos muito

utilizados para representar objetivos como comida, luz, destinos finais.

- Campo repulsivo – É o oposto ao campo atrativo, exerce uma força de repulsão sobre o robô, muito utilizado para obstáculo e predadores. Quanto mais perto o robô estiver, maior será a magnitude da repulsão, e na direção de afastamento.
- Campo tangencial – Este campo gera uma força tangencial ao objeto desejado. São forças perpendiculares as linhas radiais que se estendem para fora do objeto. São utilizados quando se deseja fazer o robô investigar um objeto ou apenas manter-se próximo a ele, mas em movimento constante tanto em sentido horário como anti-horário.
- Campo aleatório – Gera vetores aleatórios, com valores pequenos de magnitude e angulação variada (Figura 18). Muito utilizado para solucionar problemas de singularidade quando os somatórios de diferentes campos anulam a resposta desejada para uma situação causando uma colisão, por exemplo.

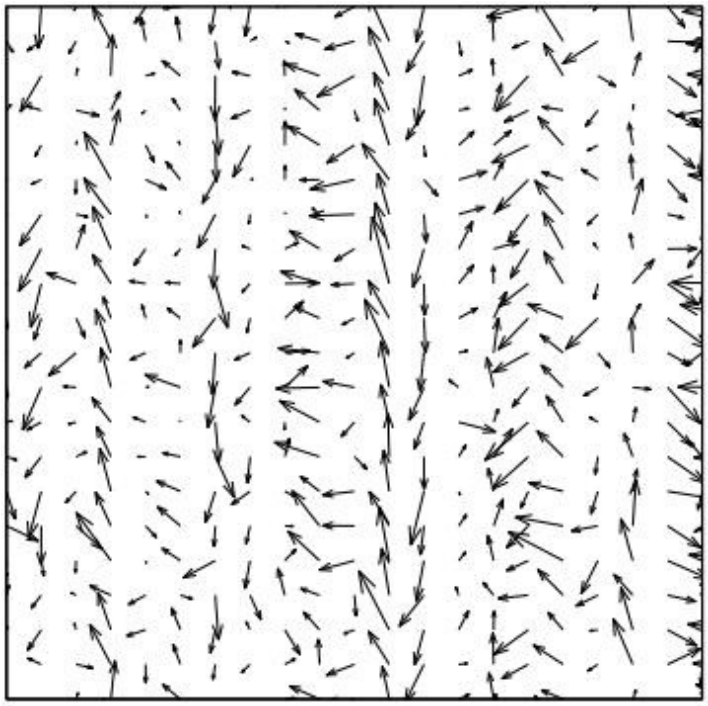


Figura 18 – Campo aleatório

Os esquemas motores clássicos utilizam os campos potenciais para a atuação motora no robô, e só calculam a contribuição de cada campo

Ao contrário dos campos onde a topologia é externamente especificada pelas condições ambientais, a topologia dos campos potenciais que um robô experimenta é determinada pelo programador. Mais especificamente, o programador atribui a cada comportamento criado uma determinada tarefa ou função, representa cada um destes comportamentos como um campo potencial, e combina todos os comportamentos para produzir o movimento do robô através da combinação dos campos potenciais mediante a superposição vetorial.

Uma representação de um campo repulsivo em três dimensões utilizada para navegação submarina de um robô móvel autônomo é apresentada na figura 19.

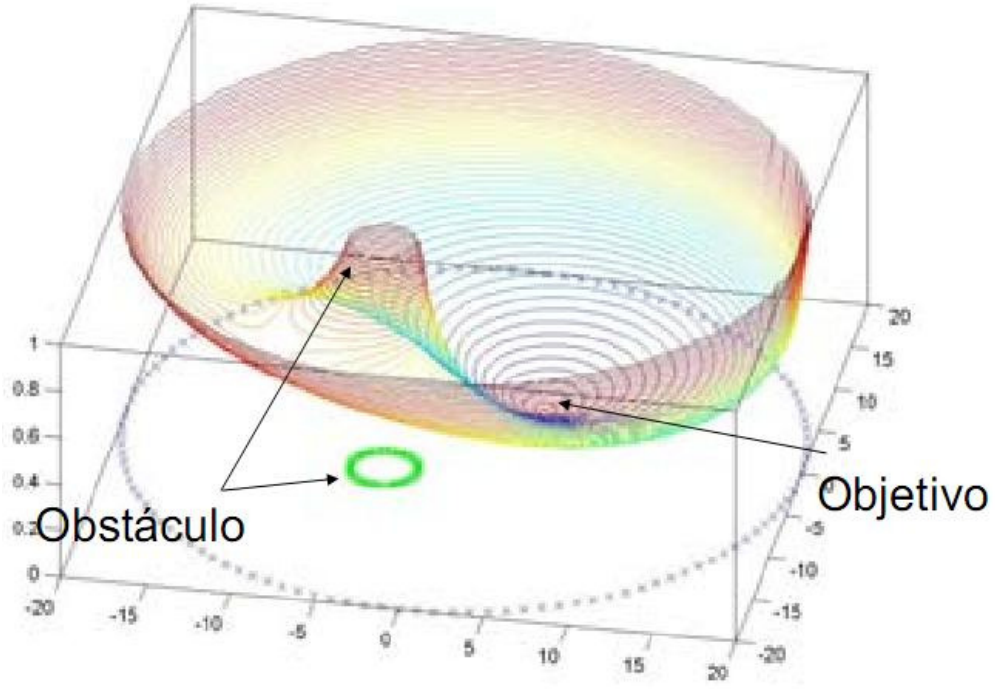


Figura 19 – Campos repulsivo 3D para navegação submarina. (MURPHY 2000) [26]

As codificações mais usadas dos campos potenciais utilizados nos esquemas motores, onde M denota a magnitude do vetor resposta, são apresentadas a seguir.

O esquema clássico *move-to-goal* (Figura 20) que define um ponto ou objeto de destino pode possuir muitas codificações de saída, como funções exponenciais decrescentes, mas descrevem-se a seguir funções simples que atendem bem às necessidades de robôs autônomos simples.

$$M = \begin{cases} \infty & \text{para } d > S \\ \dfrac{(d - R)^2}{S - R} \bullet G & \text{para } R < d \leq S \\ 0 & \text{para } d \leq R \end{cases}$$

onde a direção do vetor é radial ao redor do centro do obstáculo mas, direcionada para dentro. (Figura 20). e;

d = distância do robô até o centro do objeto.

G = ganho.

R = raio do objeto.

S = raio de interesse onde a função irá atuar.

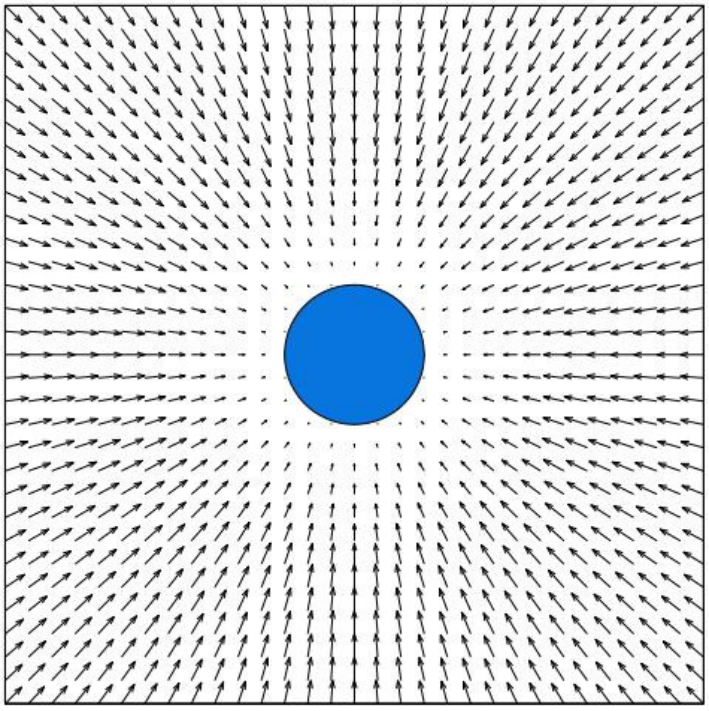


Figura 20 – Campo potencial *Move-to-goal.*

Neste esquema utilizou-se uma função simples que determina o decaimento da magnitude da velocidade do robô proporcionalmente ao quadrado da distância ao objeto. As simulações e experimentos deste trabalho utilizam robôs de pequeno porte, portando os valores para os parâmetros são expressos em cm. Os parâmetros d e S assumem valores máximos em torno de 200cm. O ganho de cada esquema pode ser atrelado a uma função que altere a capacidade de contribuição de suas saídas ao longo do processo. Entretanto, neste trabalho os

esquemas perceptivos possuem implicitamente a capacidade de contribuir com alta ou até nenhuma relevância durante o processo, por este motivo não é necessário utilizar funções que alterem os ganhos. Estes foram definidos como funções binárias onde os esquemas perceptivos são ativados e contribuem para o somatório vetorial ou simplesmente permanecem inertes.

Todos os valores referenciados de distâncias e velocidades podem ser analisados no código fonte das simulações. (Apêndice II)

Devido às limitações do ambiente de simulação foram desconsiderados os valores inerciais dos robôs. Uma vez que o foco do trabalho são robôs de pequeno porte, esta aproximação torna-se satisfatória.

O esquema *avoid-static-obstacle* é similar ao *move-to-goal*, porém gera forças de repulsão crescentes proporcionais ao quadrado da distância. (Figura 21).

$$M = \begin{cases} 0 & \text{para } d > S \\ \dfrac{S - R}{(d - R)^2} \bullet G & \text{para } R < d \leq S \\ \infty & \text{para } d \leq R \end{cases}$$

onde a direção do vetor é radial ao redor do centro do obstáculo mas, direcionada para fora. (Figura 21).

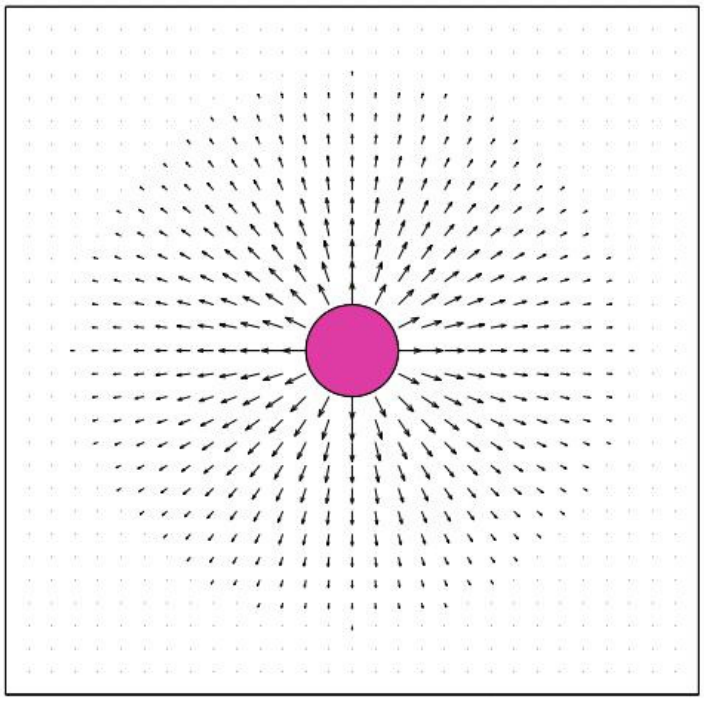


Figura 21 – Campo potencial *Avoid-static-obstacle.*

Quando se colocam ambos os objetos em um mesmo ambiente, os campos potenciais são sobrepostos e todas as forças resultantes geram o campo no qual o robô navegará.

Dentro do obstáculo, o campo de potencial repulsivo é infinito, e aponta para fora do centro do obstáculo. Fora do círculo de influência, o campo de potencial repulsivo é zero. Dentro do círculo de influência, mas fora do raio do obstáculo, a magnitude do vetor cresce proporcionalmente ao quadrado da distância do robô ao objeto.

Uma vez que haverá interações entre dois ou mais objetivos e obstáculos um campo onde todos os campos gerados se sobrepõem deve ser criado. (Figura 22).

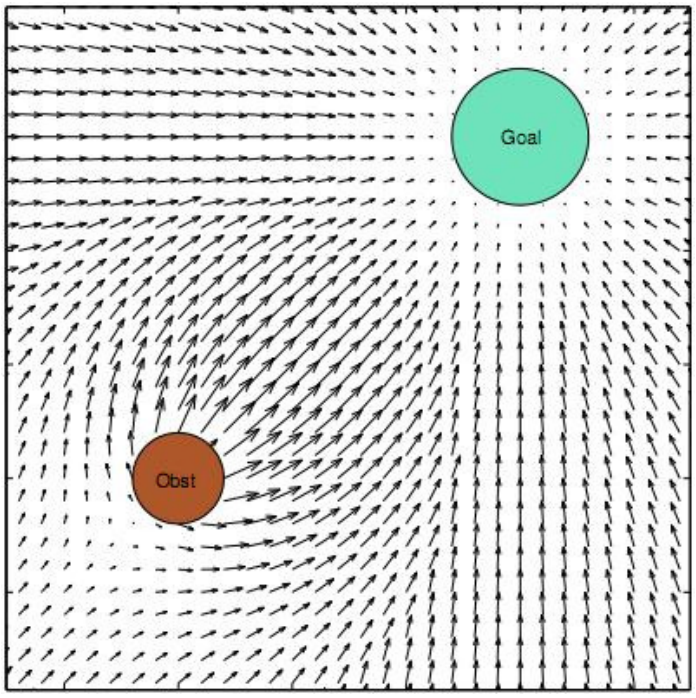


Figura 22 – Campos potenciais *Move-to-goal e Avoid-static-obstacle* sobrepostos.

A partir da criação do campo resultante final, pode-se analisar como um robô se comporta nesta situação, ou seja, qual será sua trajetória com base nos vetores resultantes da soma vetorial, vide figura 23.

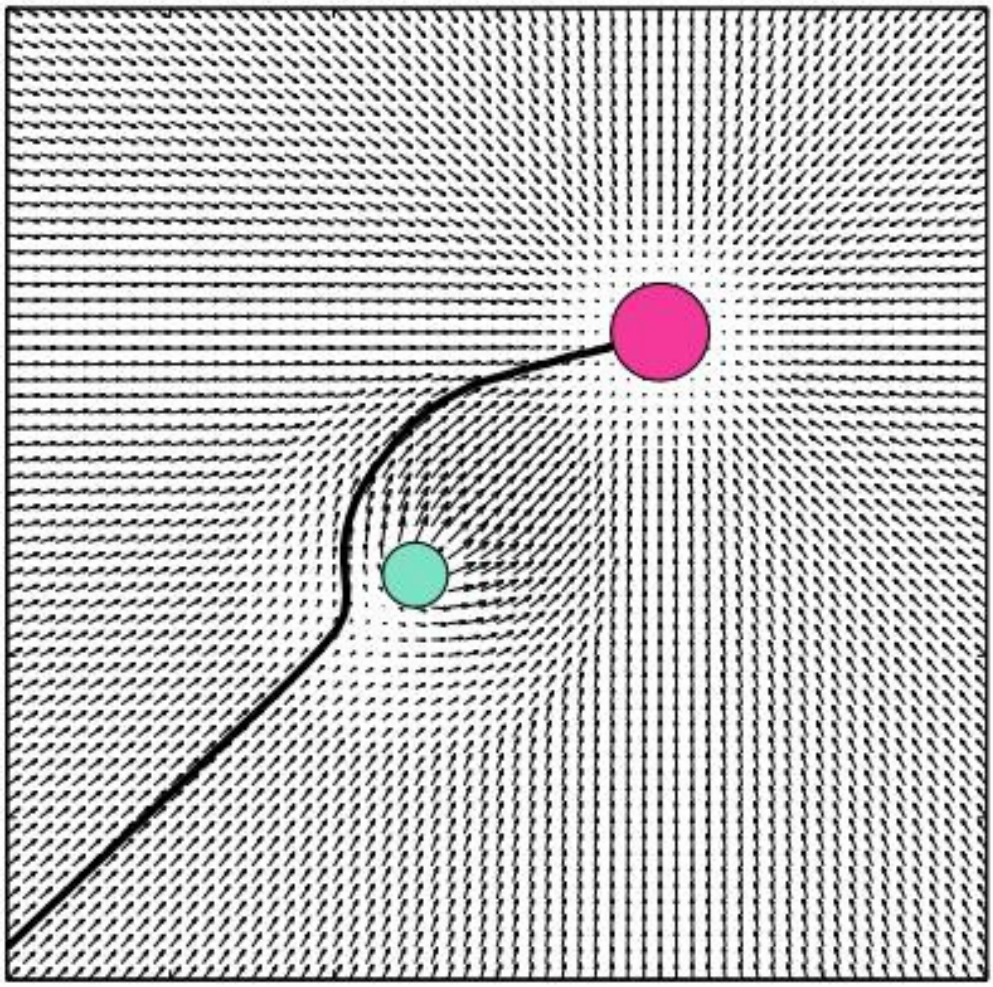


Figura 23 – Trajetória de um robô no ambiente com campos potenciais *Move-to-goal* e *Avoid-static-obstacle* sobrepostos.

Observa-se que o robô executa inicialmente uma trajetória uniforme que foi gerada pelo campo potencial do objetivo em todo o ambiente. Quando se aproxima de um obstáculo, as forças de repulsão fazem com que este seja desviado, e o somatório das forças de repulsão com as de atração do objetivo criam a trajetória com a qual o robô o alcança, desacelerando conforme a proximidade até parar completamente.

Note que existem outros tipos comuns de campos que podem ser utilizados no processo de criação do campo potencial resultante do ambiente.

2.6. Ambiente de Simulação *Player / Stage*

O *Player* é um ambiente computacional de desenvolvimento de sistemas de controle de robôs móveis amplamente utilizado por universidades, institutos de pesquisa e empresas em diversos países. Este projeto foi iniciado em 2000 por pesquisadores da *University of Southern Califórnia* para suprir a demanda de um controlador e simulador de robôs móveis, que fosse eficiente, de grande compatibilidade, e independente de arquitetura de programação do robô.

O desenvolvimento do *Player* conta com a colaboração de diversos pesquisadores das mais diversas instituições e, por ser um sistema de código aberto, de livre distribuição, e por rodar em Linux, está em constante desenvolvimento para se adequar a um número cada vez maior de plataformas robóticas e sensores comerciais. Assim como um sistema operacional cria uma interface de alto nível para facilitar o acesso de usuários e programadores ao hardware de um computador, o *Player* apresenta uma interface de acesso ao hardware de robôs móveis e sensores, facilitando sua programação e utilização. (Figura 24).

A estrutura do *Player* é baseada no modelo cliente/servidor. O servidor faz a interface com o robô e com outros sensores, obtendo dados, enviando-os para o cliente e recebendo instruções do cliente para o controle do robô e dos sensores. O cliente é o programa que controla efetivamente o robô, ou seja, a aplicação. O cliente é responsável por obter os dados do servidor, interpretá-los, e enviar instruções para o servidor, no caso o robô, para a execução de determinada tarefa.

A arquitetura cliente/servidor permite grande versatilidade no controle de robôs e sensores, de forma que um cliente pode controlar diversos servidores e diferentes clientes podem controlar diferentes sensores de um mesmo robô. Toda a comunicação entre cliente e servidor é realizada através de TCP/IP.

O cliente Player foi projetado para ser compatível com diversas linguagens e existem bibliotecas disponíveis para clientes em C, C++, Java, Python, Tcl, entre outras. Para que o cliente possa se comunicar com o servidor, é necessário que o código fonte do programa de controle inclua uma biblioteca fornecida com o Player.

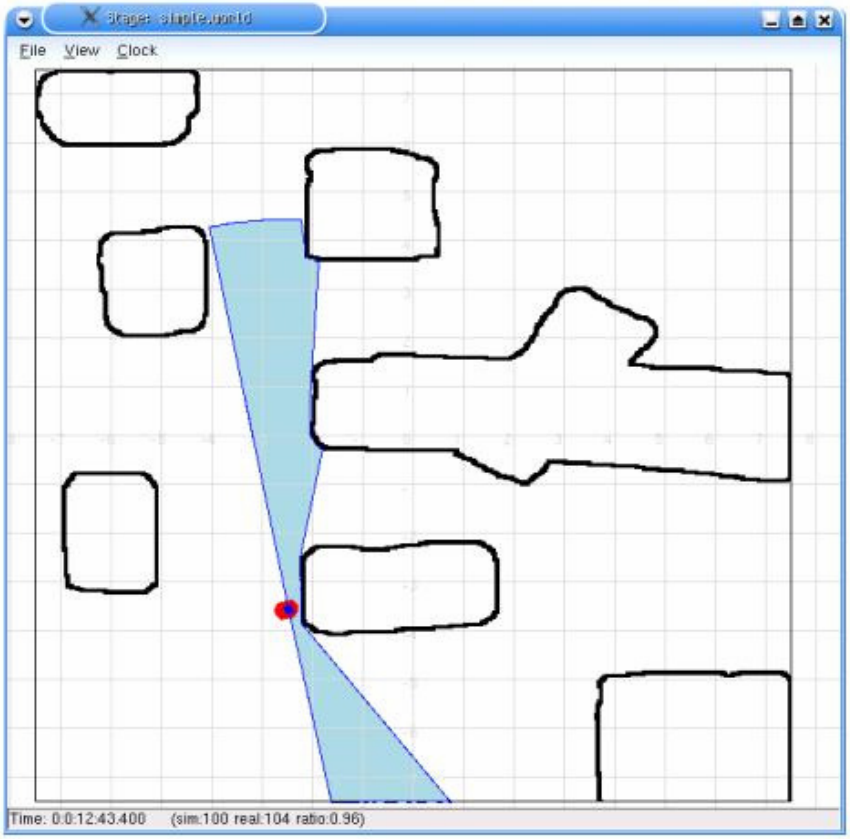


Figura 24 – Ambiente de simulação do *STAGE*.

O Stage é um módulo adicional ao Player que traz um simulador de robôs e sensores para ambientes bidimensionais. Múltiplos robôs e sensores podem ser simulados simultaneamente, controlados por um ou mais clientes. (Figura 25).

Figura 25 – Capacidade de simulação do *Stage*. (2000 robôs *Pioneer* 2-DX).

O Stage normalmente é usado para o desenvolvimento inicial de código, até que o mesmo se encontre confiável o suficiente para ser testado em robôs reais. Outras aplicações do Stage envolvem a utilização de sensores e robôs em quantidade não disponível nos experimentos, mas cujo código possa ser validado através de simulação. É também um simulador cinemático, que não leva em conta a dinâmica dos objetos simulados, contudo é possível obter grande proximidade com a realidade pelo fato de a simulação objetivar robôs de pequeno porte e rápida aceleração até atingir sua velocidade máxima. Como toda a comunicação cliente e servidor é realizada através da rede TCP/IP, é totalmente transparente para o programa cliente se o mesmo está conectado a um robô simulado ou a um robô real. Normalmente, a única modificação necessária para se executar um código desenvolvido para controlar um robô simulado em um código de um robô real é a mudança do IP do robô simulado, que é o computador que executa o simulador, para o IP do robô real.

Os projetos iniciais simulados pelo Stage estavam mais associados aos robôs *Pioneer* da empresa *ActivMedia*. Desde o início de sua implementação, a característica que mais se destaca neste ambiente de simulação é a capacidade de simular um controle de robôs, utilizando múltiplos robôs. (Figura 25).

O projeto Player/Stage avançou em relação aos projetos iniciais no sentido de permitir uma separação maior entre o hardware e o ambiente de programação dos robôs, permitindo utilizar diferentes tipos de robôs. (Figura 26).

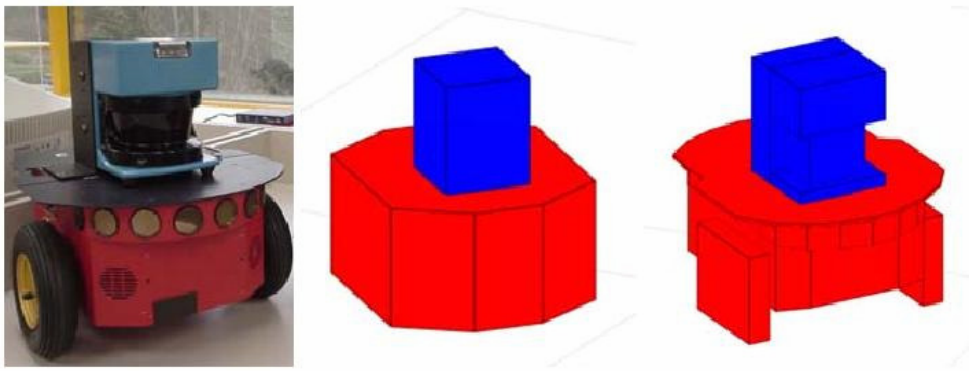


Figura 26 – Robô *Pioneer* 2-DX real, simulado simplificado e simulado de forma completa.

O projeto suporta uma série de robôs e sensores, além de uma independência de linguagem de programação, e pode simular o movimento dos

robôs e os dados coletados dos sensores a partir da utilização de um mapa em forma de uma imagem binária.

O processo de instalação e configuração do software se torna complicada em algumas ocasiões, para solucionar alguns problemas utilizou-se um guia desenvolvido por Denis Wolf (WOLF, 2009) [58].

Além do *Stage*, o *Player* também é compatível com o simulador *Gazebo*. O *Gazebo* é um simulador de robôs móveis em três dimensões que permite a simulação realista de ambientes complexos. Através do uso de bibliotecas de modelagem física, o *Gazebo* permite uma simulação extremamente fiel do comportamento e da interação física de robôs e objetos do ambiente. Por ser computacionalmente mais complexo que o *Stage*, o *Gazebo* requer mais recursos computacionais.

O *Gazebo* normalmente é utilizado em situações onde a simulação bidimensional do Stage não é fiel o suficiente. Por exemplo, em ambientes externos onde o solo é irregular e isso compromete substancialmente o funcionamento dos robôs e sensores, ou quando o robô deve mapear ambientes com objetos 3D complexos. Nesse caso, não é possível modelar esses objetos no Stage. (Figura 27).

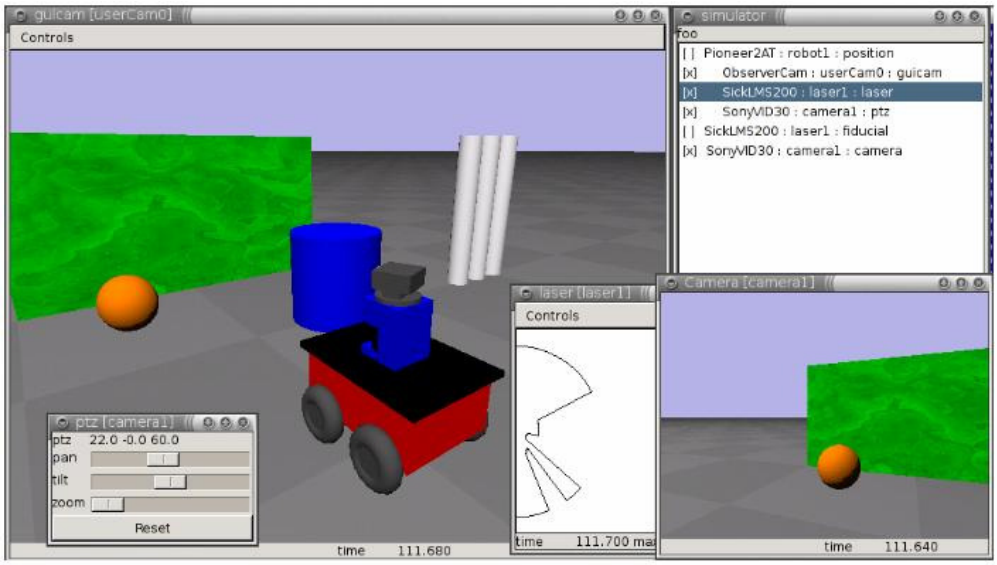


Figura 27 – Ambiente de simulação *Gazebo.*

No próximo capítulo, os sensores utilizados neste trabalho são apresentados.

3 Sensoriamento

Este capítulo apresenta a parte de sensores utilizados nas simulações e nos experimentos.

3.1. Sensoriamento da Simulação

O modelo de robô utilizado neste trabalho é o modelo clássico *pioneer 2-dx,* que possui módulos de sensoriamento que podem ser acoplados em sua simulação.

O *Stage* fornece diversos tipos de sensores que podem ser incorporados ao robô simulado. Para este trabalho foram utilizados os seguintes:

- LASER - O sensor LASER é um sensor que simula o sensor LIDAR (*Light Detection And Ranging*), que mede propriedades de luz refletida ou absorvida para obter a distância entre o robô e os obstáculos ao seu redor. O laser cobre um campo 180 graus, com 180 ou 360 leituras (leituras a cada 1 grau ou a cada meio grau), dependendo da configuração utilizada no servidor. A distância máxima que pode ser lida pelo laser é de 8 unidades de medida, em ambientes fechados. O laser pode ser configurado para a distância máxima de 80 unidades para utilização em ambientes externos. Neste trabalho é configurado conforme a necessidade do robô, e com 360 leituras, a cada meio grau. (Figura 28).

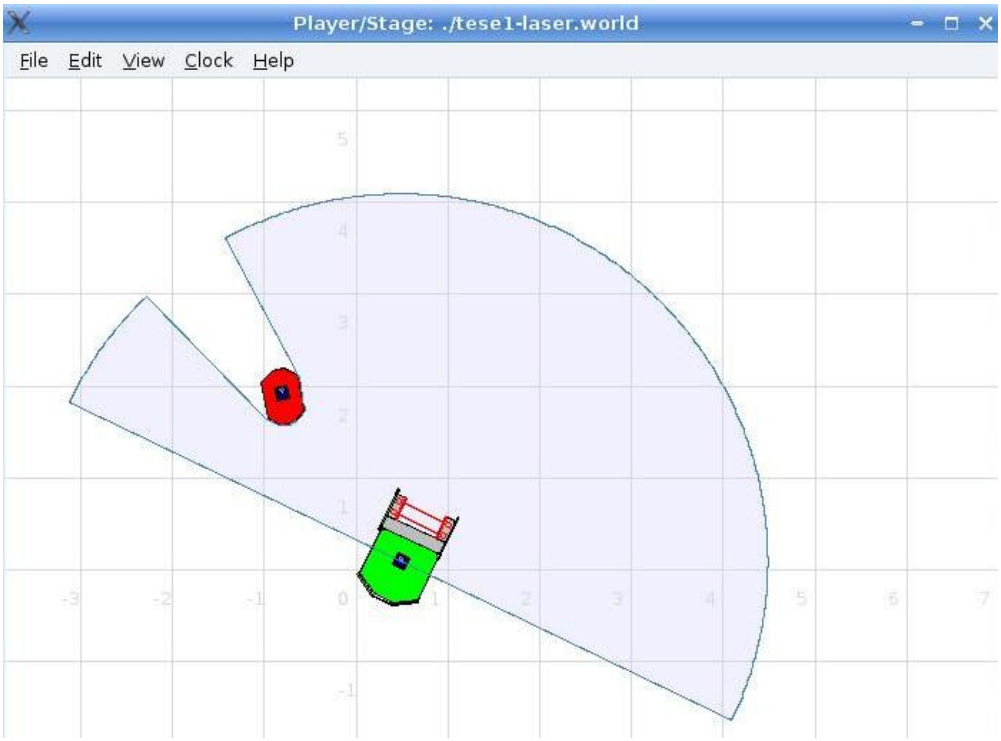


Figura 28 – Sensor LASER identificando um robô predador.

- SONAR - O sensor SONAR é um sensor que mede a distância entre o robô e os obstáculos ao seu redor, simulando sensores de ultrassom. Comparado ao laser, o sonar apresenta uma precisão menor, além de cobrir uma área menor. O número de sonares varia, dependendo do modelo do robô. Normalmente são utilizados 16 sonares apontados para direções diferentes. Como *default* no Player/Stage, a distância máxima que pode ser lida do sonar é de 5 unidades de medida. (Figura 29).

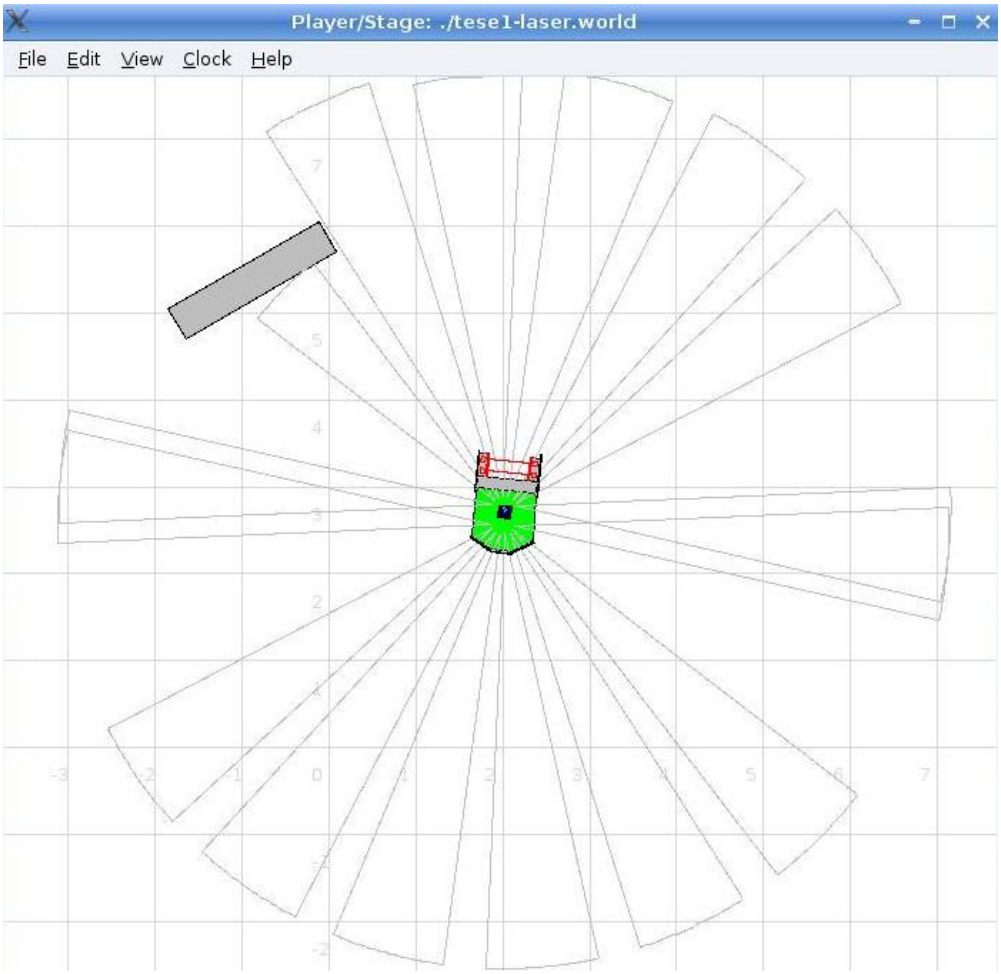


Figura 29 – Sensor SONAR identificando obstáculo.

- *BLOBFINDER* - O *blobfinder* é um dispositivo virtual que utiliza imagens de uma câmera de vídeo para localizar objetos de cores específicas, tornando mais simples tarefas como identificar ou seguir objetos de determinada cor. A simulação traz um quadro que mostra todos os objetos identificados pelo sensor e sua posição relativa em duas dimensões. Este sensor foi utilizado para detectar predadores, em vermelho, objetos desejados, em azul, e locais de destino, em amarelo. (Figura 30).

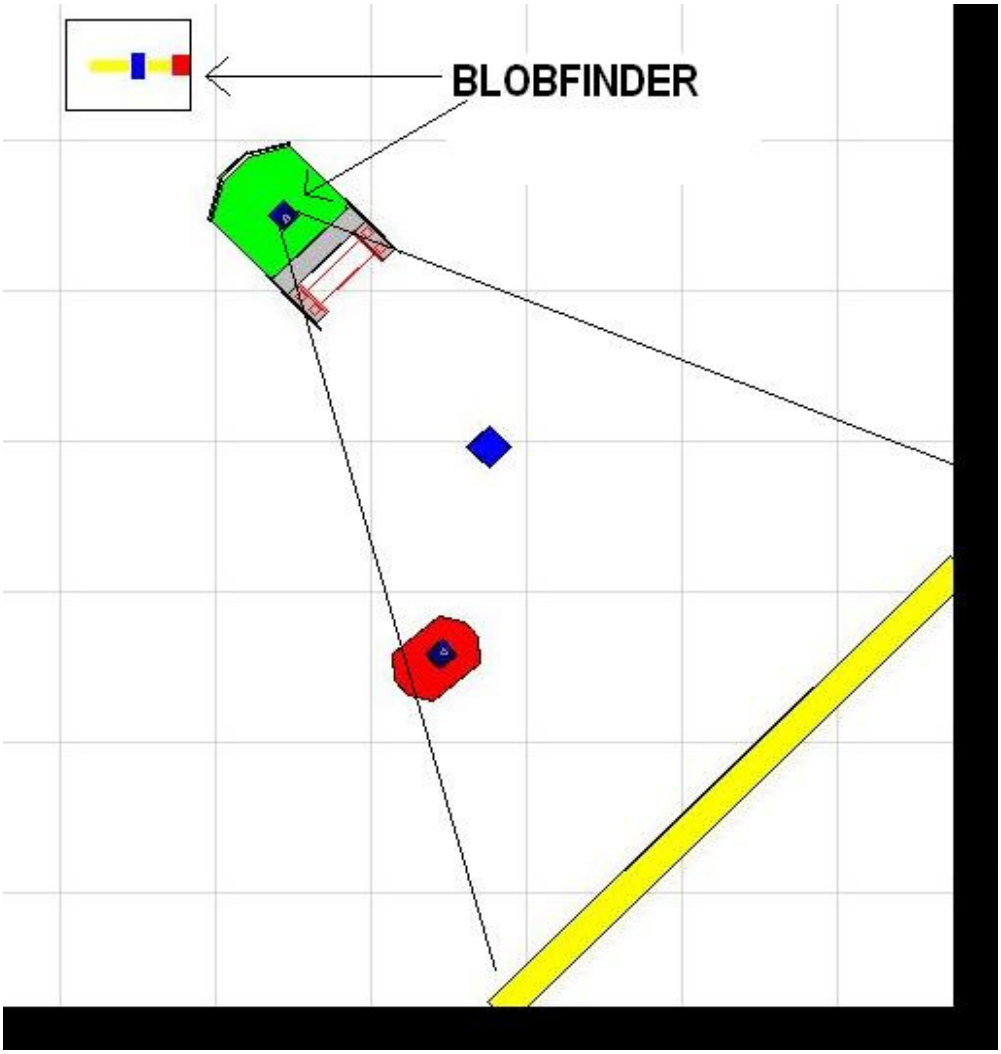


Figura 30 – Sensor *BLOBFINDER* identificando um robô predador em vermelho, objeto desejado em azul, e local de destino em amarelo.

- *GRIPPER* - O *gripper* é um conjunto de garras mecânicas verticais que pode ser usado para suspender e carregar pequenos objetos. Este possui dois sensores de presença, que são usados para garantir que um determinado objeto esteja completamente dentro da área específica para ser captado. (Figura 31).

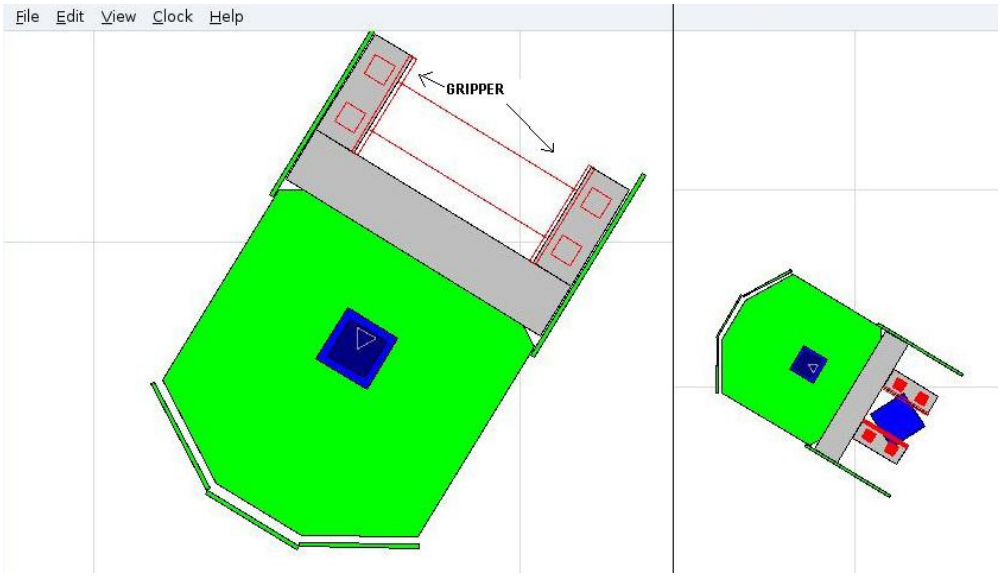


Figura 31 – *GRIPPER* com seus dois sensores de presença capturando um objeto desejado.

- *BUMPER* – O *bumper* é um sensor de colisão que funciona como um botão de dois estados, acionado ou livre. Garante que em caso de colisão o robô não fique preso, detectando-a. (Figura 32).

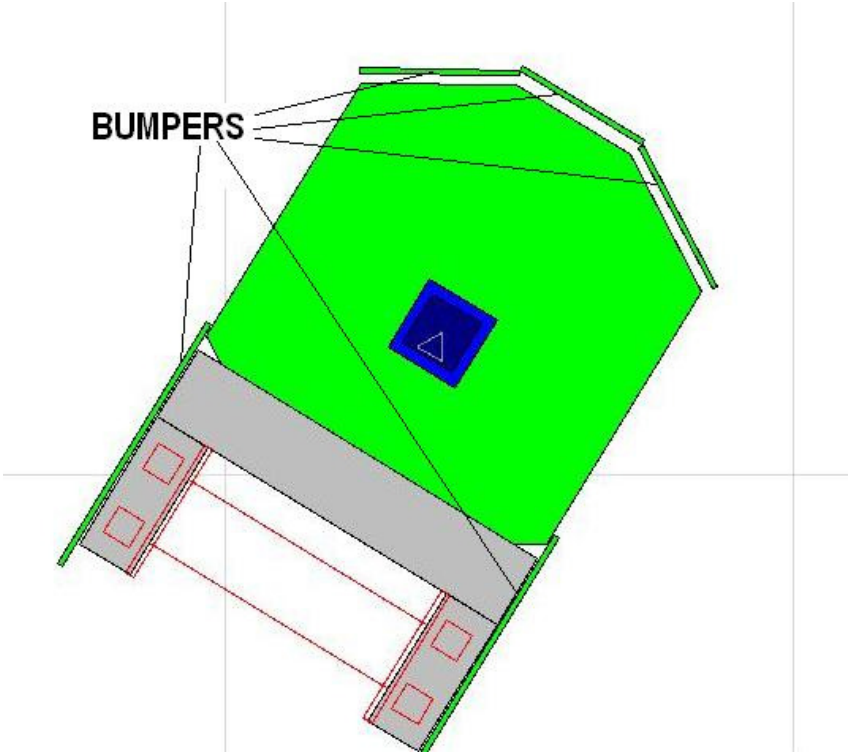


Figura 32 – *BUMPERS* Cinco bumpers dispostos ao redor do robô.

3.2. Sensoriamento do Experimento

Os robôs usados na parte experimental deste trabalho possuem a capacidade de detectar outros robôs e de identificar marcas, áreas ou linhas pintadas no solo que, neste experimento funcionam como paredes virtuais.

Os dois tipos de sensores utilizados são sensores de ultrassom e infravermelho.

Uma boa opção para detecção de outros robôs em uma arena é a utilização de sensores baseados em ultrassom. São sensores ativos, mas neste caso usam ondas de pressão, ao contrário dos infravermelhos, que funcionam por radiação eletromagnética. Estas ondas são semelhantes a ondas sonoras, contudo usam frequências mais elevadas do que o ouvido humano está preparado para detectar.

Por outro lado, como são ondas muito mais lentas, o processo de medição não é baseado na amplitude da onda recebida. Neste, calcula-se a distância ao objeto mais próximo, na direção para onde o sensor está orientado, a partir do tempo de ida e volta de um impulso sonoro.

Distâncias superiores a 2m são facilmente medidas mesmo por sensores muito básicos. São menos sensíveis ao ruído ambiente, mas têm limitações na precisão.

Visto que são baseados em ondas de pressão, todos os materiais que absorvam bem o som refletem muito pouco a onda de pressão e alteram significativamente as distâncias medidas. Por exemplo, cortinas espessas são virtualmente indetectáveis. Possuem também uma dimensão relativamente grande, o que inviabiliza a sua utilização em robôs muito pequenos. Entretanto, em robôs de dimensão média, são por vezes usados em conjunto com sensores infravermelho, aumentando bastante as capacidades de detecção de obstáculos do robô, visto que se complementam de forma bem satisfatória.

A percepção dos obstáculos estáticos e dinâmicos em um ambiente desconhecido pode ser tratada com a utilização de sensores ultrassônicos que detectam objetos num raio determinado. São fundamentais para o sucesso de qualquer tarefa em ambientes desconhecidos por possibilitarem a interação com o ambiente de forma a evitar colisões.

Existem diversos tipos de sensores ultrassônicos disponíveis para diversos alcances e utilidades diferentes, vide figura 33.

Figura 33 – Sensores ultrassônicos. [17]

Neste trabalho será utilizado um sensor ultrassônico capaz de detectar obstáculos com pelo menos um metro de distância, visto que o robô terá dimensões reduzidas.

As informações deste sensor serão adquiridas pelo micro controlador que as converterá em distâncias aos obstáculos, e com isso permitir uma movimentação sem colisões do robô.

Foram instalados 3 sensores do modelo SRF10 (Figura 34) para detectar objetos na parte frontal e nas laterais esquerda e direita.

Figura 34 – Sensor ultrassonico utilizado. Modelo SRF10.

A necessidade de evitar determinadas barreiras na superfície do ambiente a ser trabalhado faz com que o uso de sensores infravermelhos seja necessário. Um ambiente que possui descontinuidades ou linhas de cores diferentes, que servem para determinar limites ou trajetórias, no caso de robôs seguidores de linha, são exemplos práticos da utilização destes sensores.

Sensores infravermelhos são muito comuns, baratos e possuem uma rápida reposta quando se necessita verificar dois estados diferentes na superfície, como por exemplo coloração preta ou branca, o que gera uma resposta binária que é facilmente é interpretada pelo micro controlador. O sensor a ser usado neste trabalho é o QRD1114 da *FairChild Semiconductors*. (Figura 35).

Figura 35 – Sensor infravermelho utilizado. QRD1114 – *FairChild Semiconductors.* [19]

Pretende-se utilizar quatro dispositivos deste, um em cada extremidade do robô, com o intuito de evitar que o mesmo saia de um determinado espaço delimitado no plano sobre o qual se moverá, através de cores diferentes.

No próximo capítulo, as simulações desenvolvidas são descritas.

4 Simulações

Este trabalho teve três focos de tipos de simulações, com características diferentes, para gerar um processo completo na avaliação do controle baseado em comportamento.

A primeira simulação envolve um processo complexo de controle baseado em comportamento, pois engloba dez comportamentos primários que geram cinco comportamentos complexos executados em paralelo com etapas bem definidas para a execução do objetivo final com sucesso e para verificar todas as vantangens da programação baseada em comportamento.

A segunda simulação demonstra uma situação simples programada com o estilo clássico de controle, e compara a mesma situação programada utilizando o controle baseado em comportamento para verificar as diferenças entre elas.

A terceira simulação apresenta um modelo clássico de programação robótica denominado de modelo predador-presa, que foi totalmente programado baseado em comportamento e comparado experimentalmente com robôs reais.

As simulações não levaram em consideração o conhecimento prévio do ambiente e nenhum dos robôs foi programado para armazenar ações anteriores, com o intuito de tornar ainda mais ágil o processamento dos comportamentos, visto que armazenar todas as ações executadas e processar respostas melhores baseadas nesta função tornariam o processamento muito mais trabalhoso e com um uso de memória muito maior.

4.1. Simulação Completa

Como objetivo deste trabalho, a simulação se utilizará da arquitetura vertical do controle baseado em comportamento denominada esquemas motores, aliada ao uso de campos potenciais.

A tarefa definida para este trabalho foi a de coletar objetos de interesse, na cor azul, e descarregá-los em uma área de depósito, de cor amarela. Este processo consiste em desviar de obstáculos, de cor cinza, paredes, na cor preta e de outro

robô, de cor vermelha, vide figura 36. Este robô vermelho também foi programado utilizando o controle baseado em comportamento para perseguir o robô de coleta. Este, quando executa ações de fuga se comporta como um robô presa que tenta escapar de um robô predador. O robô vermelho apresenta apenas comportamentos de desviar de obstáculos e perseguir o robô verde.

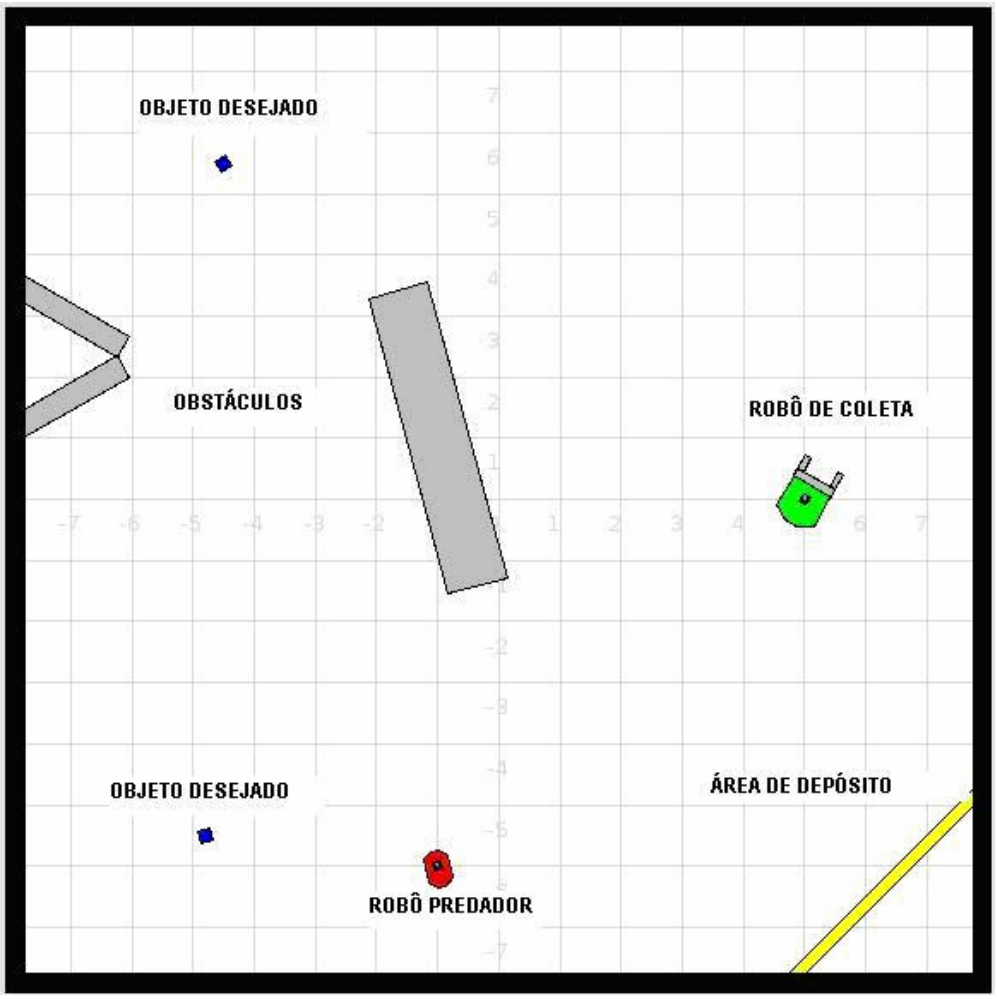


Figura 36 – Ambiente de simulação.

O ambiente de simulação Player/Stage fornece todas as ferramentas necessárias para a simulação do controle baseado em comportamento.

O modelo utilizado usa uma representação em duas dimensões do robô *Pioneer 2-dx*, muito comum em aplicações de robôs pequenos. Foram agregados a ele uma variedade de sensores capazes de perceber o ambiente de várias formas diferentes, como descrito a seguir.

Os sensores e atuadores acoplados e suas funções idealizadas são:

Sensor LASER – Cobre 180° à frente do robô com 360 pontos de medição, o que dá uma precisão angular de uma leitura a cada 0,5°. Programado com a funcionalidade exclusiva de identificar o robô predador, de cor vermelha. Seu alcance foi definido para leitura de até 4 unidades de medida.

Sensor SONAR – Total de 16 sensores dispostos ao redor do robô com a finalidade de detectar obstáculos e paredes. Configurado para uma distância máxima de 4 unidades de medida.

Sensor *BLOBFINDER* – Este sensor é responsável por detectar os objetos de interesse, cor azul, e detectar o local de despejo, na cor amarela.

GRIPPER – O *gripper* adiciona a funcionalidade de capturar e liberar os objetos de interesse.

BUMPER – Os *bumpers* sinalizam para o código do programa que houve uma colisão. Nesta simulação, existem três na parte traseira e dois na parte frontal do robô.

A partir dos sensores programados, devem-se criar os comportamentos primários, ou esquemas perceptivos que, agregados, geram os esquemas motores, ou comportamentos complexos.

Todos os comportamentos são executados simultaneamente e estão sempre contribuindo para o somatório vetorial final.

Os esquemas perceptivos definidos e suas respectivas funções são:

AndaGira – Este comportamento determina que o robô deve andar em linha reta durante um determinado tempo, depois girar em seu próprio eixo aproximadamente 270°, andar novamente em linha reta, e logo após girar no sentido oposto, ou seja, -270°. Esta instrução garante que o robô navegue cobrindo a maior parte do terreno sem retornar pelo mesmo caminho. O valor de 270º é heurístico, não foi feita nenhuma otimização relativa à exploração.

Busca – Ao identificar o objeto azul, o comportamento busca tem seu ganho elevado e começa a contribuir de forma muito significativa para o somatório geral a fim navegar o robô para próximo do objeto. É o campo potencial atrativo gerado pelo objeto que causa esta resposta.

Fechar Garra – Este comportamento só tem sua contribuição iniciada quando os sensores da garra detectam que o objeto azul se encontra posicionado

corretamente para ser capturado. Só então a garra é fechada, e logo após este comportamento deixa de contribuir.

Foge – Este comportamento contribui significativamente quando detecta o robô predador e o campo potencial repulsivo, gerado pelo mesmo, faz com que o robô se afaste do robô predador. As componentes vetoriais geradas por este comportamento variam de intensidade conforme a proximidade do predador, seguindo o equacionamento descrito para campos potenciais repulsivos.

Desvia – Tem a função de desviar dos obstáculo estáticos do ambiente como as paredes e os objetos cinza dispostos no mundo simulado. Este comportamento também utiliza o campo potencial repulsivo para desviar dos obstáculos, e tem constantemente sua contribuição alterada durante a execução. Pela grande quantidade de sensores e obstáculos existentes, o robô está constantemente se acomodando da melhor maneira possível para evitar obstáculos estáticos.

Colisão – Este comportamento foi programado para detectar colisões iminentes, ou seja, quando os sensores de proximidade detectarem que algum objeto, tanto estático quanto dinâmico, está a menos de 1,5 unidades de medida; sua contribuição se torna alta para tentar evitar ao máximo que haja alguma colisão.

Ruído – O comportamento ruído está constantemente contribuindo com uma pequena parcela na soma vetorial, apenas para garantir que o robô escape de situações singulares onde 2 campos potenciais que estejam atuando sobre o robô tenham forças exatamente opostas e se anulem. Com este comportamento, pode-se garantir que algum dos campos irá prevalecer, fornecendo alguma orientação sobre o robô.

Entrega – Uma vez que o objeto azul foi capturado pela garra e o local de depósito foi detectado, este comportamento é responsável por contribuir para que o robô navegue diretamente para ele.

Abrir Garra – Este comportamento só tem sua contribuição iniciada quando o robô se encontra dentro da área de depósito do objeto azul. Só então a garra é aberta para liberação do objeto. Após a liberação, a garra retorna à posição aberta, o robô gira 180° em seu próprio eixo, e então este comportamento deixa de contribuir.

Batida – Quando efetivamente ocorrer uma colisão indicada por algum dos sensores do tipo *bumper*, todas as contribuições dos outros comportamentos têm seu ganhos zerados para que este comportamento de emergência possa recuperar a trajetória do robô. Este processo acontece muito rapidamente, exatamente para representar a reação de girar para o lado oposto de onde ocorreu a colisão.

Todos os comportamentos definidos até agora são esquemas perceptivos puramente reativos, mas não conseguiriam executar a tarefa por completo se não houvesse o complemento da teoria do controle baseado em comportamento que orienta a criação dos esquemas motores, ou comportamentos complexos e um arbitrador central para determinar quais comportamentos complexos devem ser ativados em cada momento. Este arbitrador, em sistemas com dezenas de comportamentos, pode alternar entre os esquemas motores utilizando funções lineares ou não lineares, porém neste trabalho executar esta troca de forma binária e sequencial trouxe resultados satisfatórios, tornando a programação mais simples.

Os esquemas motores estão definidos da seguinte forma:

- **EXPLORAR:**

Este é o esquema motor inicial quando o robô se encontra no ambiente e não está detectando nem está carregando o objeto azul. Este comportamento complexo é gerado pela contribuição dos comportamentos AndaGira, Foge, Desvia, Colisão, Ruído e Batida. Ou seja, o robô está explorando o ambiente em busca do objeto desejado, mas com as funcionalidades que os comportamentos citados lhe fornecem. Apenas estes esquemas perceptivos contribuem para a soma vetorial quando este comportamento é ativado pelo arbitrador.

- **AQUISITAR:**

O arbitrador só aciona este comportamento quando o robô está com o comportamento EXPLORAR ativado e detecta um objeto azul. Isto determina que o robô não esteja mais explorando o ambiente em busca do objeto, e sim que deve entrar no processo de aquisitá-lo. É a junção dos esquemas perceptivos Busca, Foge, Desvia, Colisão, Ruído e Batida.

- **PEGAR:**

Assim que o objeto azul entra na área de captação da garra, o arbitrador ativa este comportamento. Ele é composto apenas pelos esquemas perceptivos Fechar Garra e Foge, pois no momento em que ocorre a captação do objeto pela

garra o robô está parado, portanto não precisa desviar de nenhum obstáculo nem se preocupar com colisões, apenas em caso de aproximação do robô predador é que ocorrerá a soma vetorial.

- **ENTREGAR:**

É similar ao comportamento EXPLORAR, porém ao invés de estar explorando o ambiente à procura do objeto azul, procura-se a área de depósito amarela. Possui os comportamentos simples Entrega, Foge, Desvia, Colisão e Batida. Este esquema motor é ativado assim que a garra termina de capturar o objeto azul.

- **SOLTAR:**

Assim que o robô entra na faixa de depósito, o arbitrador ativa este comportamento, que tem como única finalidade abrir a garra para despejar o objeto azul na zona amarela, e logo após girar 180° em seu próprio eixo. Após este giro, o arbitrador ativa novamente o esquema motor EXPLORAR, fechando o ciclo de organização dos comportamentos complexos. Apenas os comportamentos simples Abrir Garra e Foge contribuem no somatório vetorial geral enquanto este esquema motor está ativo.

A combinação e codificação dos esquemas perceptivos e motores, assim como o funcionamento do arbitrador podem ser visto no apêndice II.

A descrição do controle do robô predador é uma simplificação do robô de coleta, pois este não possui as tarefas de captura e entrega de objetos, ele apenas é programado para explorar e perseguir o robô de coleta. Portanto, o robô predador apresenta apenas os esquemas perceptivos AndaGira, Desvia, Colisão, Ruído e Batida. Os esquemas motores gerados a partir destes esquemas perceptivos são o EXPLORAR, que na essência é o mesmo comportamento do robô de coleta, mas buscando o robô verde ao invés do objeto azul, e o esquema motor PERSEGUIR, que se assemelha ao comportamento AQUISITAR. Contudo se diferencia no objetivo a ser alcançado e também no campo potencial gerado, que no objeto azul é um campo atrativo decrescente com a proximidade do objeto e, no robô verde, um campo atrativo uniforme, tentando maximizar a velocidade da perseguição para aumentar as chances de cumprir o objetivo.

As figuras 37 e 38 apresentam a topologia dos esquemas perceptivos e motores, e sua organização frente ao arbitrador de cada robô.

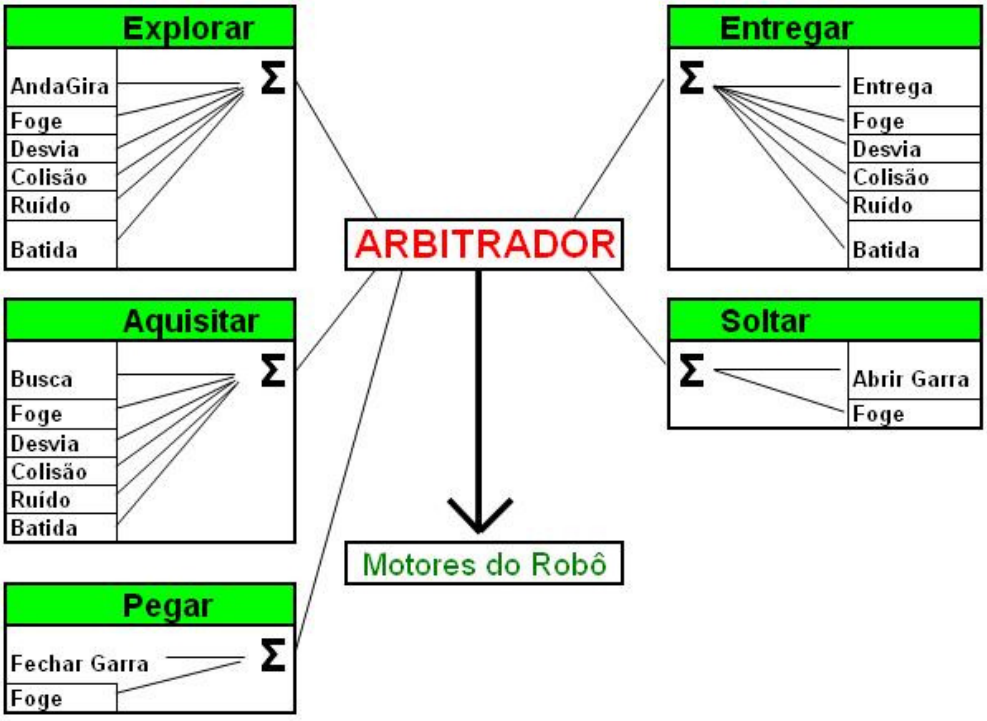


Figura 37 – Topologia dos esquemas motores e perceptivos do robô coleta.

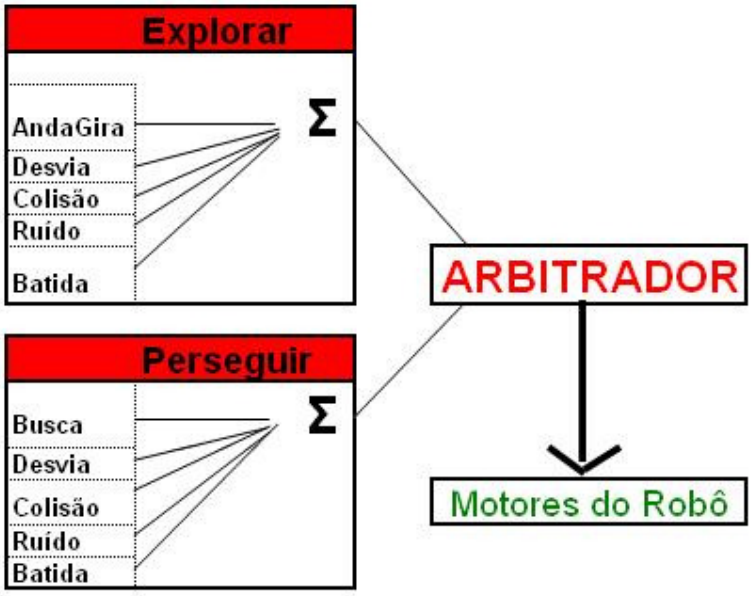


Figura 38 – Topologia dos esquemas motores e perceptivos do robô predador.

O player/stage trabalha com unidades adimensionais para os parâmetros dos robôs. As velocidades máximas do robô de coleta variam entre -0,7 (para trás)

e 1 (em frente) e do robô predador variando -0,3 (para trás) e 0,61 (em frente). A velocidade angular de ambos foi definida entre -5 (para esqueda) e 5 (para a direita). O início das simulações mostrou que as equações que geram os campos potenciais conseguem executar todas as tarefas de maneira satisfatória por isso, os ganhos relativos aos esquemas perceptivos foram definidos de forma binária, 0 e 1. Desta forma o arbitrador determina quais esquemas perceptivos estarão ativos em cada esquema motor. Esta configuração torna a programação mais transparente, pois é possível identificar de forma clara qual comportamento é o predominante em cada momento da simulação.

Exemplos de trajetórias do robô de coleta, para cada esquema motor em separado, são apresentadas a seguir. (Figura 39).

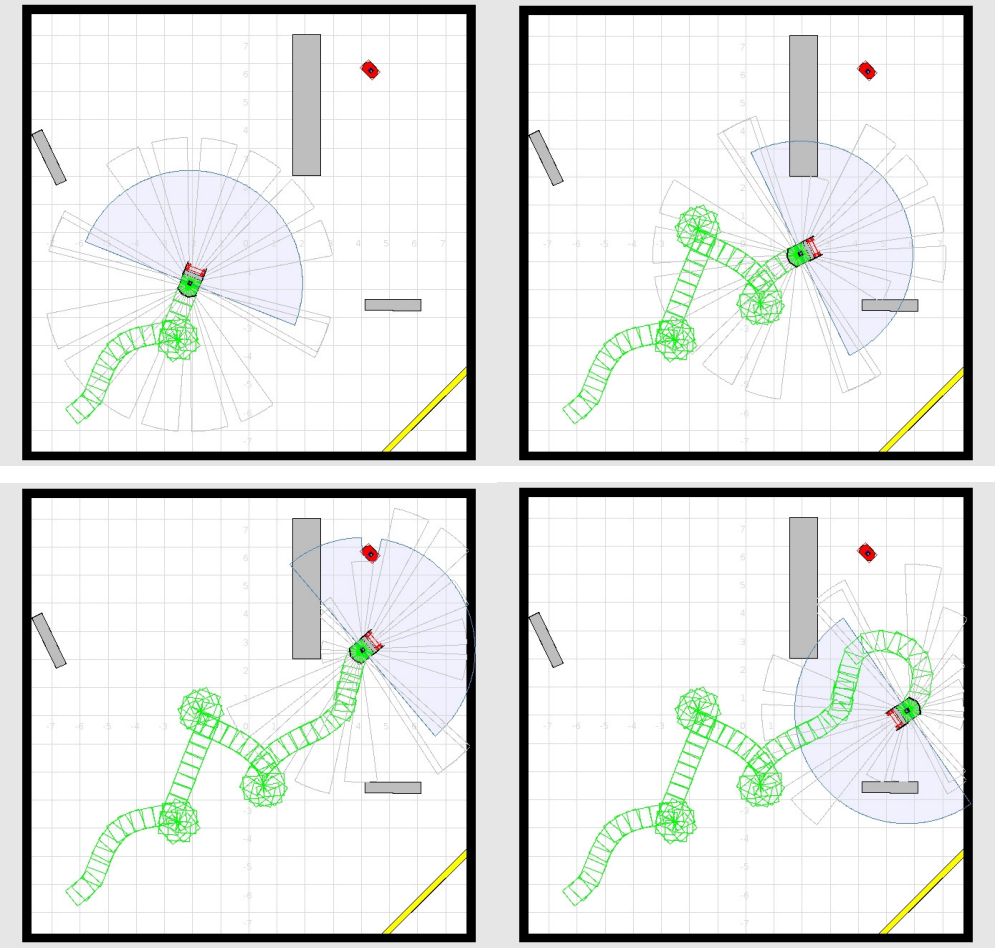


Figura 39 – Trajetória executada pelo robô de coleta apenas com o comportamento EXPLORAR ativo.

Na figura 39, o robô de coleta, em sua situação inicial tem o esquema motor EXPLORAR ativo e inicia o processo de exploração do ambiente à procura do objeto azul, com os outros esquemas perceptivos sempre contribuindo para evitar colisões e fugir do predador. A situação representada na figura 39 não possui os objetos azuis para garantir que apenas o comportamento EXPLORAR esteja ativo. Inicialmente o robô se encontra na parte inferior esquerda do ambiente e começa a se locomover. Logo no inicio da locomoção já é possível ver que o esquema perceptivo Desvia está contribuindo significativamente, orientando o robô para a melhor trajetória que se afaste das paredes. É uma reação ao campo potencial repulsivo das paredes.

Assim que o processo é iniciado, todos os esquemas perceptivos agregados ao esquema motor EXPLORAR contribuem para a locomoção do robô, e logo após o inicio da trajetória o esquema perceptivo AndaGira orienta o robô para girar 270° em seu próprio eixo para verificar se existe algum objeto desejado ou predador ao seu redor.

A segunda parte da figura mostra que a trajetória do robô já explorou mais da metade da área do mapa e está caminhando para uma área onde há um robô predador.

Assim que o robô detecta o robô predador através do sensor LASER, ou seja, assim que o robô vermelho entra na faixa de detecção do sensor, o comportamento foge contribui com grande importância para a soma vetorial e se torna praticamente dominante. Caso esta contribuição não fosse alta, o campo potencial do obstáculo cinza, à esquerda, e da parede, à direita, faria com que o robô fosse diretamente de encontro ao predador. Ou seja, o campo potencial repulsivo do robô predador gera componentes vetoriais de alta magnitude e fazem com que o robô desvie corretamente, tomando uma trajetória contrária como visto na última etapa.

O robô inimigo, chamado aqui de predador, não é considerado um membro nocivo nesta simulação, ou seja, ele é apenas um obstáculo dinâmico a ser evitado. Uma analogia para esta escolha seria o robô de coleta ser um robô de escritório e o robô predador representar uma pessoa caminhando. O robô de coleta não tem como objetivo se desvencilhar do robô inimigo e sim desviar para evitar colisões. Comportamentos no qual um robô deve atacar e o outro fugir são discutidos na simulação predador/presa.

O robô predador tem seu alcance de visão propositalmente maior que o do robô de coleta e sua velocidade menor, para agregar diferenças significativas entre ambos.

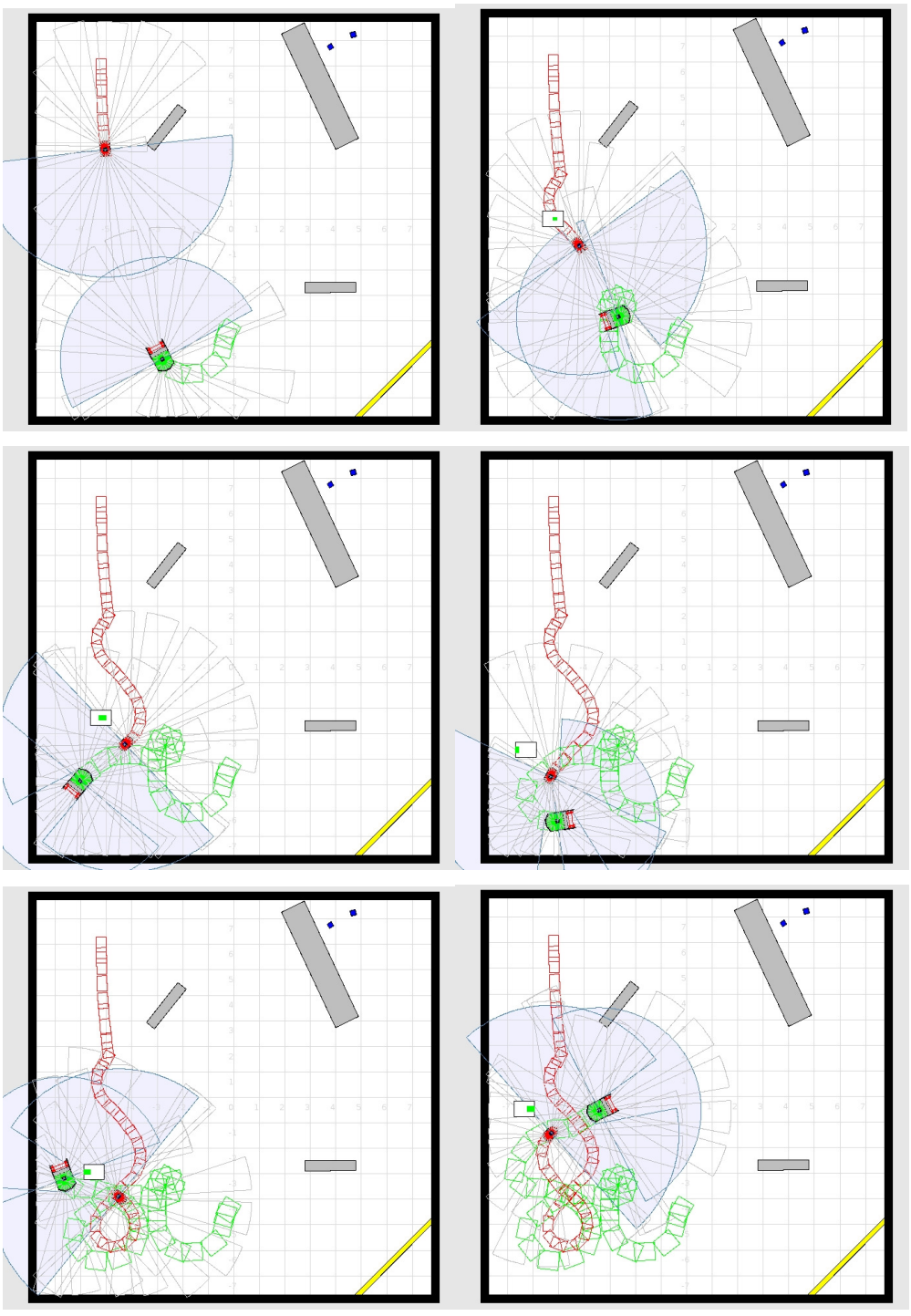


Figura 40 – Trajetória de desvio do robô predador feita pelo robô de coleta.

Na trajetória da figura 40, os robôs estão em rota de encontro a partir do momento em que o robô de coleta entra na faixa de sensoriamento do robô predador. Automaticamente o esquema motor PERSEGUIR é ativado pelo arbitrador do robô predador, fazendo com que o campo atrativo gerado pelo robô de coleta exerça forças diretamente para sua posição. Até este momento, o robô de coleta não identificou o predador por ter um alcance menor em seus sensores, e prossegue explorando sem contribuições do esquema perceptivo FOGE.

Após esta parte, começa o processo no qual os robôs se identificam e só então o robô de coleta inicia uma trajetória para desviar do robô predador, agregando as contribuições dos esquemas perceptivos ativos.

Nas quatro partes restantes da figura 40 nota-se o robô de coleta sendo guiado pelo somatório dos campos potenciais das paredes próximas e do forte campo repulsivo gerado pelo robô predador, até conseguir ir para uma área aberta.

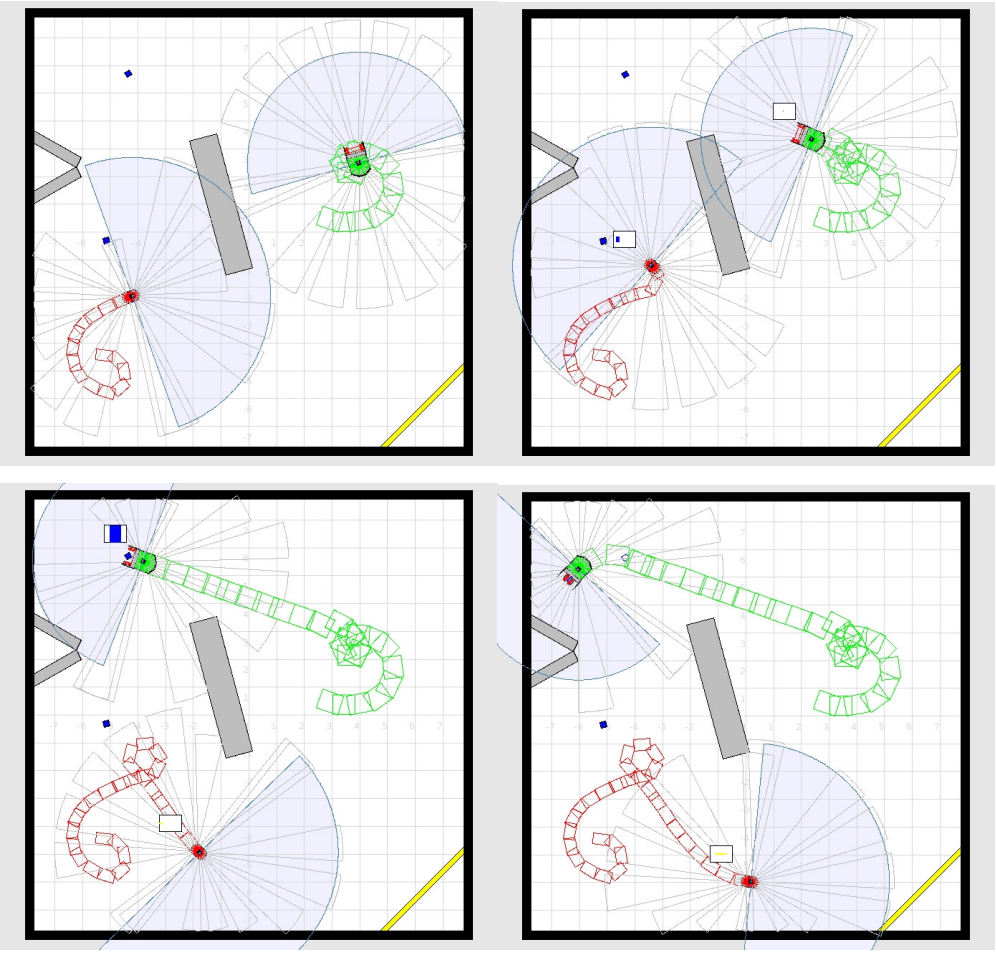


Figura 41 – Trajetória dos esquemas motores AQUISITAR, PEGAR e a ativação do ENTREGAR.

Na figura 41, o robô de coleta está executando o giro em torno de seu eixo explorando o ambiente, até que o sensor *BLOBFINDER* detecta a presença do objeto desejado e a trajetória circular é interrompida imediatamente, pois neste momento o arbitrador ativa o esquema motor AQUISITAR. O objeto desejado agora exerce um campo potencial atrativo decrescente sobre o robô, gerando uma trajetória praticamente linear em direção ao objeto. O termo AQUISITAR é sinônimo do verbo adquirir e amplamente utilizado no meio técnico.

Percebe-se que a influência dos esquemas perceptivos de desvio e contra colisões tem sua contribuição diminuída, pois o robô passa bem próximo ao obstáculo cinza e não sofre alteração significativa em sua trajetória. Esta é uma característica projetada, pois o objetivo principal da simulação é o objeto azul e, com isso, o campo potencial gerado pelo objeto azul é tão forte que os vetores que fariam o robô desviar não têm magnitude suficiente para mudar a trajetória do robô, a não ser que este entre em rota de colisão com algum obstáculo ou predador. Neste caso, como as equações que estabelecem os campos potenciais repulsivos geram magnitudes inversamente proporcionais ao quadrado da distância, em proximidades pequenas os vetores de desvio corrigiriam a trajetória do robô.

Uma vez que o campo potencial atrativo do objeto azul conduz o robô a parar com o objeto dentro da área de captação de sua garra, o arbitrador ativa o esquema motor PEGAR. Neste comportamento, o robô fecha a garra para poder carregar o objeto, e a única contribuição possível neste momento é do esquema perceptivo Foge, pois não há necessidade de desviar de obstáculos visto que o robô está parado.

Ao fechar a garra, tem-se a ativação do esquema motor ENTREGAR, e se inicia a navegação em busca do local para depósito.

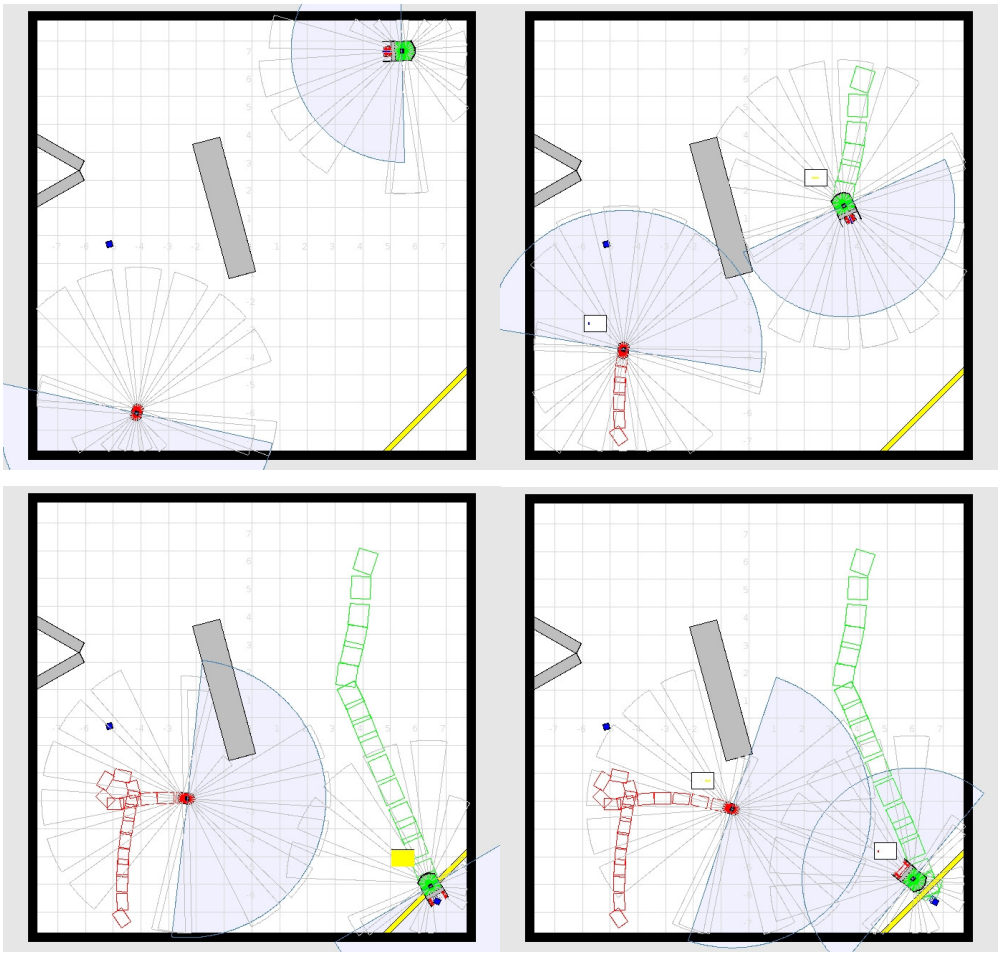


Figura 42 – Trajetória dos esquemas motores ENTREGAR, SOLTAR e a ativação do EXPLORAR.

O processo ENTREGAR procura pela área de depósito como visto na figura 42, e assim que a detecta entra em uma trajetória similar à descrita no comportamento AQUISITAR, onde os campos potenciais de desvio, fuga e contra colisões só influem de maneira significativa em casos iminentes de colisão. Desta maneira, o robô ao detectar a zona amarela é imediatamente atraído pelo campo potencial atrativo decrescente gerado por ela e, ao entrar na zona de depósito, o arbitrador aciona o esquema motor SOLTAR. Este esquema tem como finalidade abrir a garra para depositar o objeto azul e logo após girar 180° sob seu eixo para sair da zona amarela. Logo após esta ação, o arbitrador torna a ativar o esquema motor EXPLORAR, reiniciando o ciclo a procura de outro objeto desejado.

Todos os esquemas motores desta simulação executaram suas tarefas com êxito, o robô alcançou o objetivo mesmo com as dificuldades neste ambiente.

4.2. Comparativo Clássico X Comportamento

A estratégia do controle baseado em comportamento se mostra muito útil na programação de robôs móveis autônomos, contudo faz-se necessária uma comparação com a técnica tradicional de programação que não leva em consideração os modelos comportamentais e utiliza a arquitetura horizontal para executar as tarefas.

Nesta simulação, utilizaram-se os mesmo dois robôs da simulação anterior, porém o objetivo do robô de coleta, de cor verde, foi reduzido para apenas navegar pelo ambiente desviando de obstáculos e do robô vermelho. O objetivo do robô vermelho também não foi modificado, permanecendo explorar e perseguir o robô verde.

4.2.1. Controle Baseado em Comportamento

A parte da simulação programada utilizando o controle baseado em comportamento contempla um único esquema motor NAVEGAR, com esquemas perceptivos AndaGira, Foge, Desvia, Ruído e Batida.

A programação do robô vermelho continua com dois esquemas motores EXPLORAR e PERSEGUIR com seus respectivos esquemas perceptivos, vide figura 43.

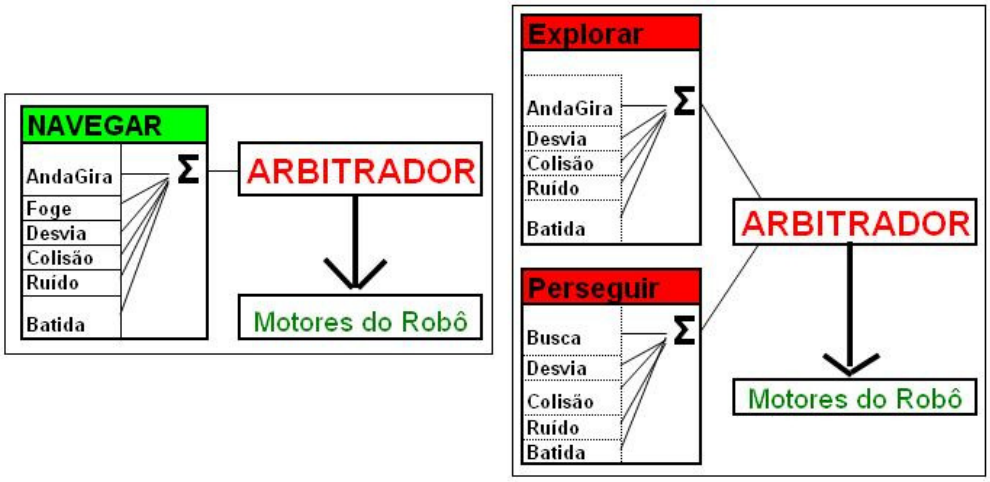


Figura 43 – Topologia do robô verde e do robô vermelho programados com controle baseado em comportamento.

A figura 44 apresenta o comportamento que os robôs apresentam em uma simulação simplificada.

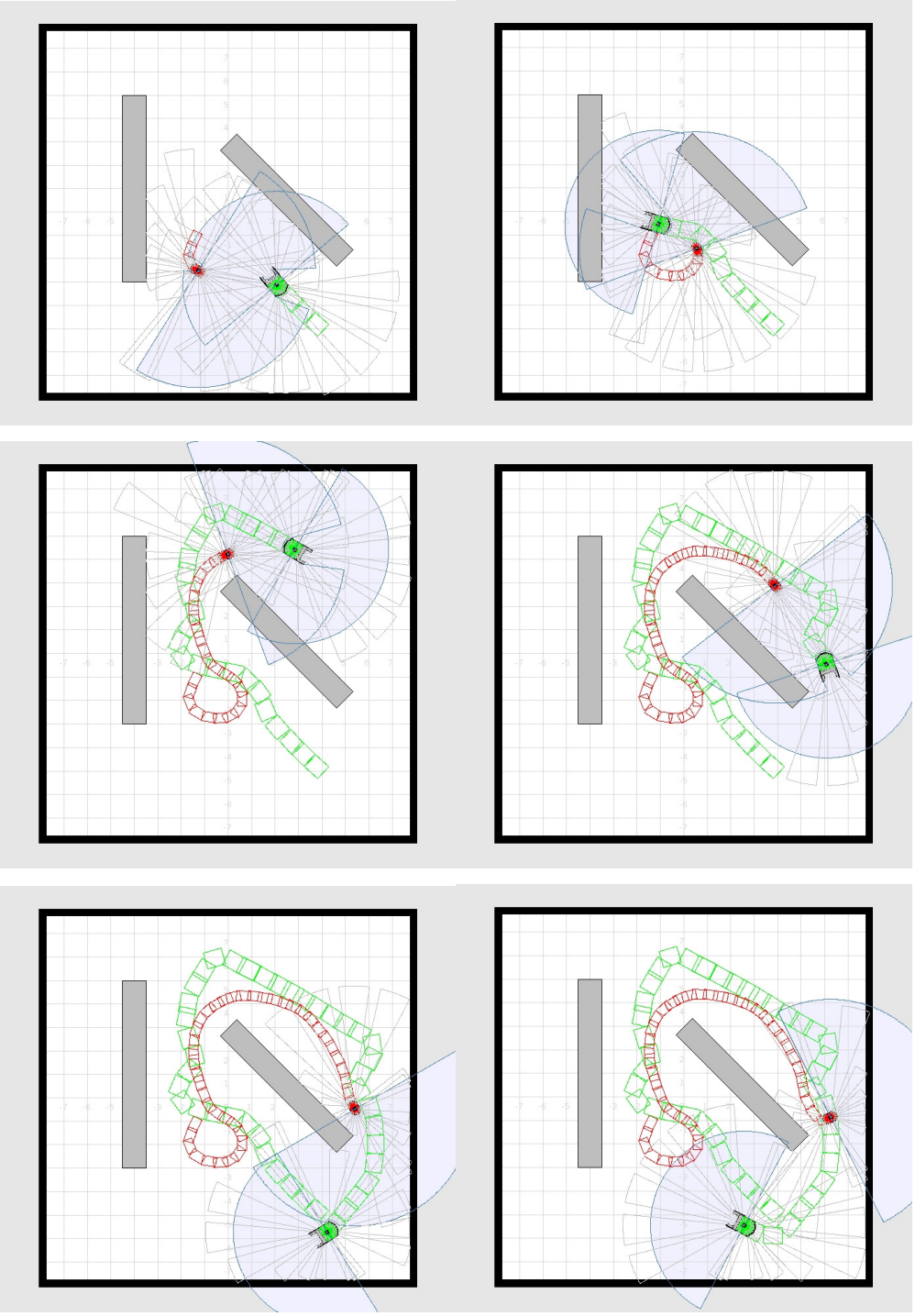


Figura 44 – Trajetória dos robôs controlados por comportamento. Robô verde desvia e escapa do robô vermelho.

Esta simulação (Figura 44) demonstra o bom desempenho do controle baseado em comportamento, utilizando campos potenciais, que foram responsáveis pela correta trajetória do robô verde ao desviar dos obstáculos estáticos e dinâmicos do ambiente.

O estado inicial mostra que os robôs estão em rota de colisão e o campo potencial repulsivo crescente gerado pelo robô vermelho gera forças vetoriais que forçam o correto desvio por parte do robô verde. Após a detecção do robô verde, o robô vermelho tem seu esquema motor PERSEGUIR ativado pelo arbitrador, e inicia o processo de perseguição por todo o ambiente. Apesar de possuir sensores com maior alcance, o robô vermelho é mais lento e por isso, o robô verde consegue sair do alcance do vermelho, apesar da influência dos obstáculos.

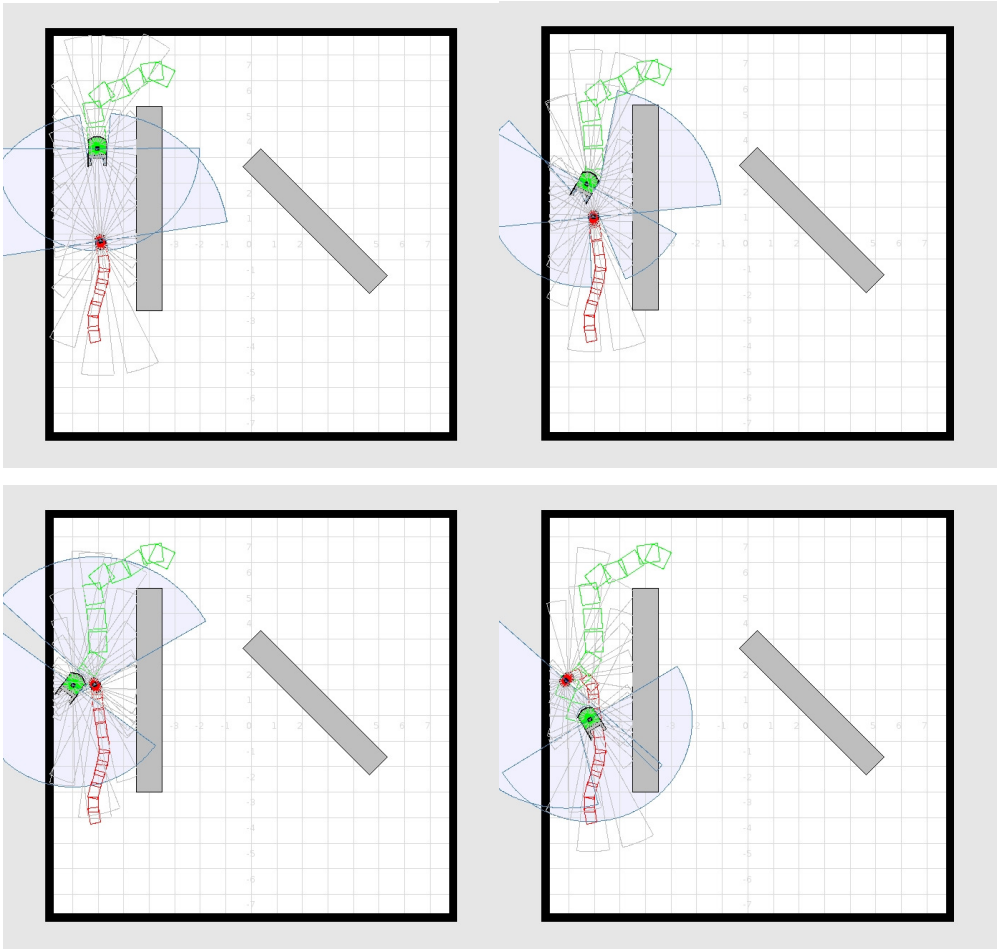


Figura 45 – Trajetória dos robôs verde e vermelho (baseados em comportamento) em rota de colisão em um corredor estreito.

Na figura 45, um corredor estreito é utilizado para verificar a trajetória realizada pelos robôs. Assim que entram no corredor, ambos os robôs sofrem as mesmas forças geradas pelos campos potenciais das paredes, e admitem uma trajetória linear e igualmente afastada de ambos os lados. Esta trajetória gerada cria uma rota de colisão entre os robôs. Cabe ao robô verde a tarefa de tentar desviar do robô vermelho.

O campo potencial repulsivo crescente criado pelo robô vermelho começa a influenciar o robô verde, que pouco altera sua rota, pois está sofrendo as forças laterais das paredes. O campo repulsivo cresce inversamente com o quadrado da distância entre os robôs, ou seja, quando o robô verde se aproxima muito do vermelho, as forças repulsivas tornam-se maiores que as forças das paredes, forçando o desvio.

Uma observação sobre o campo repulsivo é que a magnitude da velocidade também é uma grandeza que cresce inversamente com o quadrado da distância. Este fato é um dos responsáveis para o sucesso na tarefa de desviar do robô vermelho.

Uma possível interpretação de fuga seria fazer com que o robô verde girasse 180º assim que avistasse o robô vermelho. Este fato não acontece pois a simulação foi concebida com objetivo de capturar objetos e ao mesmo tempo desviar de obstáculos. Ainda neste caso o campo potencial repulsivo tende a girar o robô verde mas o campo repulsivo das paredes é dominante e logo após o desvio os campos potenciais das paredes retomam a rota centralizada de ambos.

4.2.2. Controle Clássico de Trajetórias

A fim de realizar a comparação, as duas programações, baseada em comportamento e clássica, foram totalmente refeitas utilizando a teoria clássica de programação horizontal. Esta divide o controle em unidades de função dependentes umas das outras, onde as informações captadas pelos sensores são utilizadas para representar o ambiente a cada instante. A partir desta representação, ocorre o planejamento da ação a ser executada para atingir seu objetivo. O planejamento é executado em tempo real e utilizado para navegar o robô até encerrar o ciclo de controle e, logo após, reiniciá-lo.

Nesta simulação com controle clássico não foi utilizado o conceito de campos potenciais visto que sua implementação em um sistema onde as funções são dependentes umas das outras dificulta o processo de somatório dos campos vetoriais.

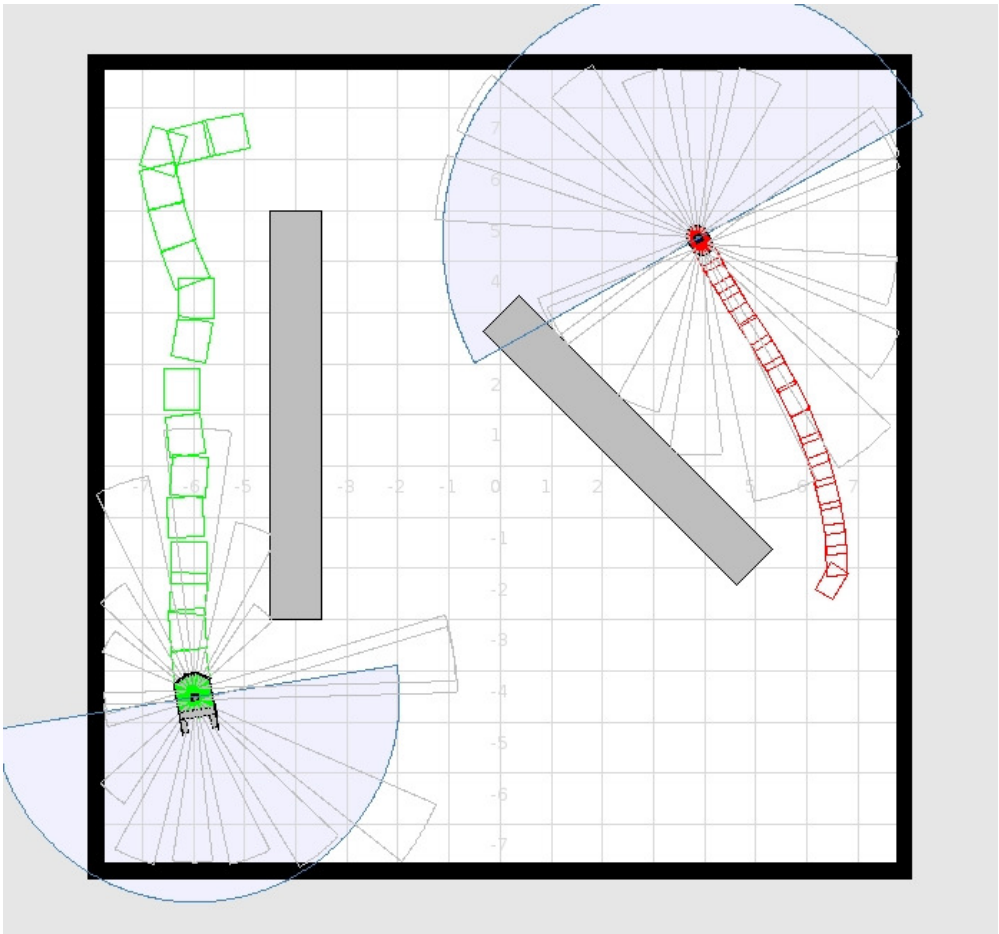


Figura 46 – Trajetória não linear do robô verde dentro do corredor.

A reprogramação dos robôs utilizando o método clássico trouxe características diferentes.

Primeiramente, vê-se na figura 46 que o robô verde não consegue executar uma trajetória perfeitamente linear quando se encontra sozinho dentro de um corredor estreito. Este fato se deve ao estilo da programação clássica, que executa módulos de função separadamente e sequencialmente; isto significa que quando o robô entra no corredor, o módulo responsável pelo desvio detecta que o obstáculo à direita é o que está mais próximo, sendo assim executa ordem para o robô girar à esquerda. Esta ordem de execução faz com que o robô gire para a esquerda e comece a navegar para longe do obstáculo à direita, porém, logo depois que esse

processo se inicia, o obstáculo à esquerda se torna o mais próximo ao robô, fazendo com que o módulo de desvio dê a ordem inversa.

Esta troca de operações faz com que o robô não obedeça a uma trajetória tão retilínea quanto à vista no controle utilizando campos potenciais.

Este problema poderia ser solucionado realizando um ajuste fino para cada situação específica que o robô vier a enfrentar, porém isto demanda mais gasto computacional, tempo de programação e aumento da complexidade do código gerado.

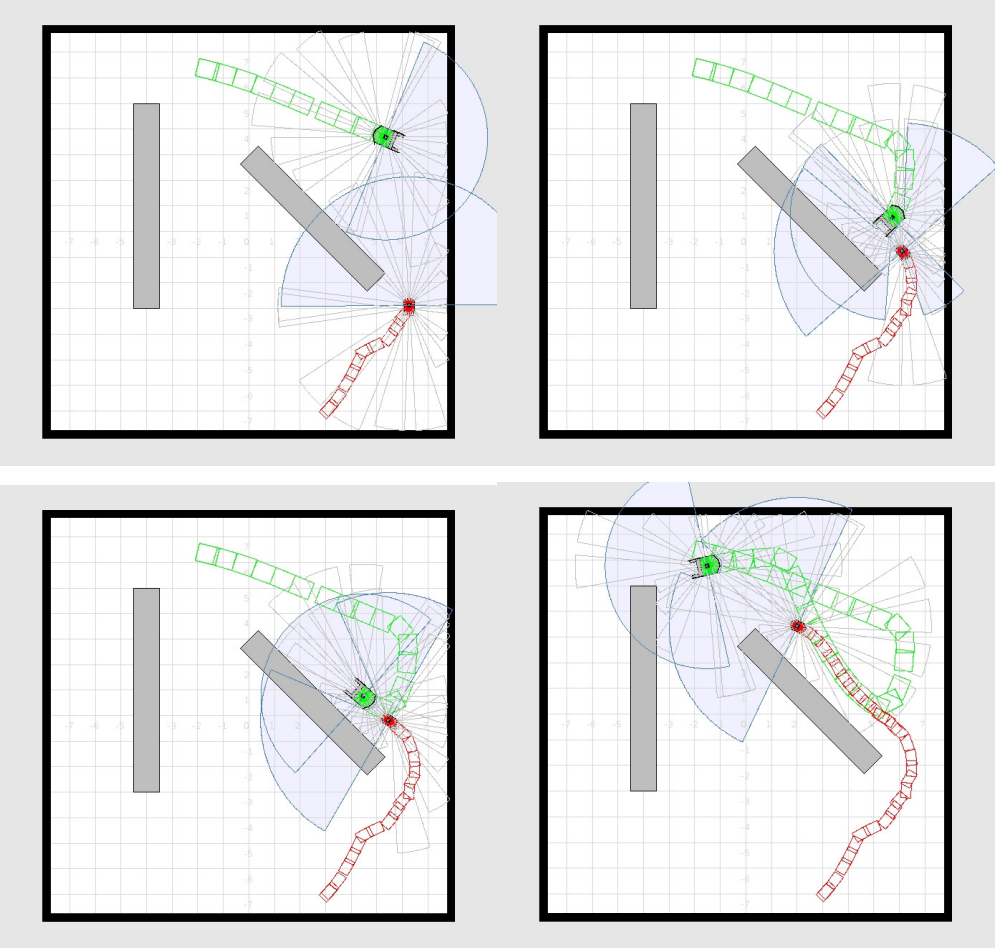


Figura 47 – Trajetória de desvio dos robôs programados com método clássico.

As curvas dos robôs são mais bruscas (Figura 47) pelo fato de não haver a soma vetorial repulsiva e atrativa. Esta característica não impede seu funcionamento, mas perde-se na fluidez do movimento como se vê,

provavelmente gerando maior gasto de energia em um sistema real. Ao aproximar-se da parede o módulo responsável pelo desvio executa a ordem de giro assim que detecta a parede, isso faz com que o robô gire rapidamente 90° para evitar a colisão, e segue em frente.

À frente, ao detectar o robô vermelho, o módulo de fuga executa a ordem de giro, que nesta situação é no mesmo sentido do desvio do obstáculo que está próximo. Neste caso, a tarefa de desvio foi realizada com sucesso, mas com movimento abruptos que em experimentos reais podem causar incertezas de medições.

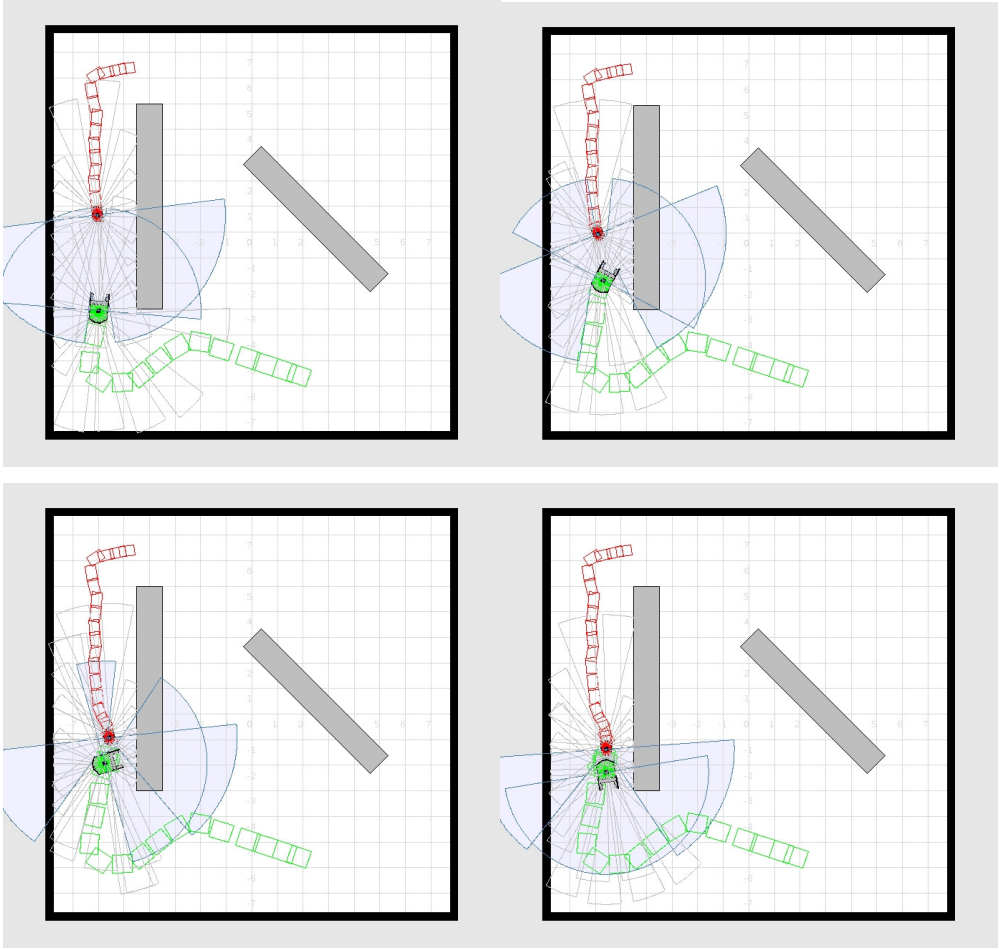


Figura 48 – Colisão entre os robôs programados com método clássico.

A mesma situação do corredor estreito é aplicada ao controle clássico (Figura 48). A aproximação do robô verde ocorre de maneira não retilínea como

descrito anteriormente, este fato contribui para a incapacidade de realizar o desvio.

O módulo responsável pelo desvio das paredes opera normalmente até que o robô vermelho é detectado. Neste momento, o módulo responsável por fugir atua sobre os motores para que girem à direta com grande magnitude. Logo após esta ordem, o módulo responsável por desviar das paredes também atua no robô para afastá-lo da parede, contudo como no modelo clássico não existe o método dos campos potenciais crescentes, a intensidade do giro dos motores para fugir do robô vermelho é maior do que a de desviar das paredes. Isto causa uma colisão com as paredes e consequentemente com o robô vermelho, até que depois de três colisões o robô consegue girar e tomar o caminho contrário.

4.2.3. Comparação

A programação clássica apresentou desvantagens significativas em relação ao controle baseado em comportamento aliado aos campos potenciais:

A interdependência dos módulos de funções do controle clássico deixa o código "amarrado" em relação à sua execução. Os módulos são executados sequencialmente, gerando respostas conflitantes que podem causar incertezas nos experimentos reais. Perde-se a estrutura modular. Caso seja necessário adicionar um novo módulo de função, toda a estrutura do código fonte deve ser alterada, causando retrabalho desnecessário.

No controle por comportamento, todos os módulos comportamentais são executados em paralelo, o que traz alta modularidade. É possível adicionar um novo comportamento sem praticamente nenhuma alteração no código original.

As derivadas de primeira e segunda ordem da posição com relação ao tempo fornecem, respectivamente, a velocidade e a aceleração do objeto. Menos conhecida é a derivada terceira da posição com relação ao tempo, isto é, a taxa de variação da aceleração com relação ao tempo que, em inglês, recebe o nome de *jerk*. Esta derivada é útil, por exemplo, quando se deseja estudar o desgaste pela ação do movimento de um mecanismo sensível ou o desconforto dos passageiros dentro de um veículo. A ausência dos campos potenciais deixa a trajetória dos robôs grosseira, com movimentos bruscos, ou seja, com valores altos da terceira

derivada da posição, causados pelo conflito de respostas diferentes originadas pelos módulos de função.

Esta ausência causa problemas mais frequentes de colisões e indecisões sobre qual melhor direção seguir por parte do robô.

O advento dos campos potenciais trouxe suavidade para o controle do robô, visto que apenas uma força resultante atua no robô o tempo todo, gerada a partir do somatório vetorial de todos os esquemas perceptivos ativos no momento.

4.3. Predador / Presa

Na ecologia, a interação entre diferentes espécies é foco de estudo de diversas áreas da ciência. O modelo predador / presa, no qual um ou mais agentes do tipo predador tem como objetivo caçar um ou mais agentes do tipo presa, é discutido como o problema predador / presa. Este problema é naturalmente visto como uma tarefa cooperativa entre agentes. A estratégia mais eficaz para a solução exige comportamentos cooperativos entre os agentes individuais. O modelo predador/presa foi proposto por Benda (BENDA, 1986) [44] e se tornou referência para pesquisadores avaliarem as técnicas de comportamentos cooperativos.

Neste trabalho, a simulação realizada contempla a perseguição de um robô presa por um robô predador. A programação baseada em comportamento se encaixa no escopo desta tarefa por reproduzir o comportamento animal que pode ser verificado experimentalmente. Esta simulação é a representação de um modelo real existente. Dois robôs identicamente projetados foram programados de acordo com a teoria do controle baseado em comportamento com o método dos campos potenciais, e utilizando o modelo predador / presa simulados.

Os robôs possuem três sensores ultrassom com a finalidade de detecção do outro robô, sendo que o robô predador possui os sensores localizados a esquerda, direta, e à frente, mas o robô presa tem o terceiro sensor posicionado na parte traseira com o intuito de poder detectar a aproximação do robô predador e poder fugir da melhor maneira possível. Caso o ultrassom fosse instalado na parte da frente, o robô presa não teria como fugir eficientemente. Uma vez que este execute um giro no próprio eixo para iniciar a fuga, o robô predador sairia do

ângulo de visão da presa, fazendo com que o robô presa, que não possui memória, acredite que não está mais sendo perseguido.

Existem também quatro sensores infravermelhos para detecção de superfícies brancas. Localizados nas extremidades da base do robô, estes sensores estão apontados para o solo, para detectar regiões e linhas brancas que delimitam o ambiente.

O ambiente simulado contempla um robô predador programado para detectar e perseguir o robô presa, que por sua vez foi programado para detectar e fugir do robô predador. Existem faixas largas nas bordas do ambiente que representam áreas de risco, ou seja, paredes virtuais das quais o robô deve desviar assim que as detectar. O campo potencial gerado pelas paredes virtuais é um campo repulsivo constante; como a largura da faixa de desvio é pequena, o robô deve desviar de maneira rápida para não sair da arena com nenhuma de suas rodas. Ambos têm a mesma capacidade de desvio, contudo o robô presa foi programado para ser mais rápido que o robô predador, pois os comportamentos de fuga e desvio deixam o robô presa muito vulnerável a ser encurralado. A proporção do quanto a velocidade do robô presa é superior foi obtida empiricamente para tentar representar melhor o comportamento animal no qual o predador nem sempre obtém sucesso ao perseguir uma presa.

Os comportamentos definidos em cada robô seguem a teoria do controle baseado em comportamento, utilizando a arquitetura dos esquemas motores e esquemas perceptivos. Os dois robôs diferem no esquema motor CAÇAR do predador e do FUGIR da presa, mais especificamente no esquema perceptivo Busca do predador e Foge da presa. Todos os esquemas são variações dos esquemas utilizados na simulação completa e, detalhados a seguir:

AndaGira – Determina que o robô deve andar em linha reta durante um determinado tempo, depois girar em seu próprio eixo aproximadamente 90°, andar novamente em linha reta, e logo após girar no sentido oposto, ou seja, -90°. Como o terreno tem dimensões reduzidas, esta instrução garante que o robô explore uma área considerável em pouco tempo.

Busca – Presente apenas no robô predador. Ao identificar o robô presa, tem seu ganho elevado e começa a contribuir de forma muito significativa para o somatório geral a fim de navegar o robô para próximo do objeto. É o campo potencial atrativo gerado pelo objeto que causa esta resposta.

Foge – Presente apenas no robô presa. Contribui com alta relevância quando detecta o robô predador e o campo potencial repulsivo gerado pelo mesmo faz com que o robô se afaste do robô predador. As componentes vetoriais geradas por este comportamento variam de intensidade conforme a proximidade do predador, seguindo o equacionamento descrito para campos potenciais repulsivos crescentes.

Desvia – É um comportamento associado a cada um dos quatro sensores infravermelhos, que contribui para o somatório quando cada um deles é ativado individualmente. Tem a função de desviar das paredes virtuais representadas pelas faixas nas bordas do chão da arena. Este comportamento também utiliza o campo potencial repulsivo para desviar dos obstáculos. A presença do robô posicionado com apenas um sensor infravermelho detectando a presença da faixa de desvio indica que o robô deve efetuar um leve desvio, por isso a contribuição deste comportamento tem sua magnitude reduzida. À medida que o robô entra mais na faixa de desvio, outros sensores detectam a faixa branca, acionando outros esquemas perceptivos e contribuindo cada vez mais para o desvio.

Colisão – Este comportamento foi programado para detectar quando o robô entre de frente ou de costas na faixa de desvio, ou seja, quando os dois sensores frontais ou traseiros estão acionados ao mesmo tempo, e os outros dois estão desativados. Nesta situação o campo potencial repulsivo contribui com grande intensidade para que o robô recue e gire, ou avance e gire, para evitar uma invasão por completo na faixa de desvio.

Batida – Quando três sensores infravermelhos estiverem acionados simultaneamente, significa que o robô está em uma situação crítica, ou seja, está posicionado quase que por completo dentro da faixa de desvio. O campo potencial repulsivo neste caso contribui significativamente para navegar o robô de volta para a arena.

Ruído – O esquema perceptivo ruído está constantemente contribuindo com uma pequena parcela na soma vetorial, apenas para garantir que o robô escape de situações singulares onde dois campos potenciais que estejam atuando sobre o robô tenham forças exatamente opostas e se anulem. Com este comportamento, pode-se garantir que algum dos campos irá prevalecer, fornecendo alguma orientação sobre o robô.

Os esquemas motores estão definidos da seguinte forma: (Figura 49).

- **EXPLORAR:**

Este é o esquema motor inicial quando o robô se encontra no ambiente e não está detectando o outro robô. Este comportamento complexo é gerado pela contribuição dos comportamentos AndaGira, Desvia, Colisão, Ruído e Batida. O robô explora o ambiente com as funcionalidades que os comportamentos citados lhe fornecem. Apenas estes esquemas perceptivos contribuem para a soma vetorial quando este comportamento é ativado pelo arbitrador.

- **CAÇAR:**

Presente apenas no robô predador. O arbitrador só aciona este comportamento quando o robô está com o comportamento EXPLORAR ativado e o robô presa à frente ou nas laterais. Isto determina que o robô passe do estado exploratório e passa a ter o objetivo de caçar a presa. Compõem-se dos esquemas perceptivos Busca, Desvia, Colisão, Ruído e Batida.

- **FUGIR:**

Presente apenas no robô presa. O arbitrador só aciona este comportamento quando o robô está com o comportamento EXPLORAR ativado e o robô predador na parte traseira ou nas laterais. Isto determina que o robô passe do estado exploratório a ter o objetivo de fugir do predador, se posicionando de maneira que o robô predador fique diretamente atrás, a fim de atingir a maior velocidade possível de afastamento. Compõem-se dos esquemas perceptivos Busca, Desvia, Colisão, Ruído e Batida.

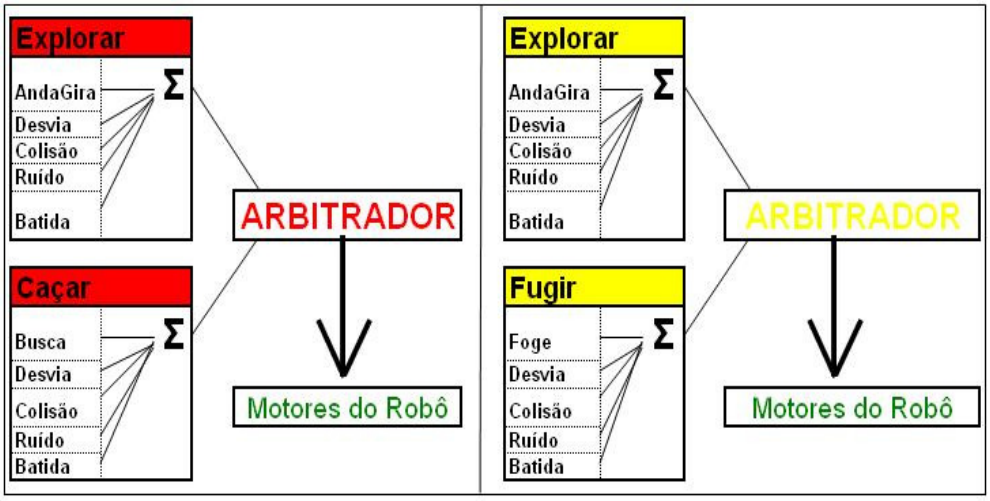


Figura 49 – Topologia do robô predador e do robô presa programados com controle baseado em comportamento.

As simulações apresentam o comportamento dos robôs interagindo na área retangular que também é usada no experimento real. (Figura 50).

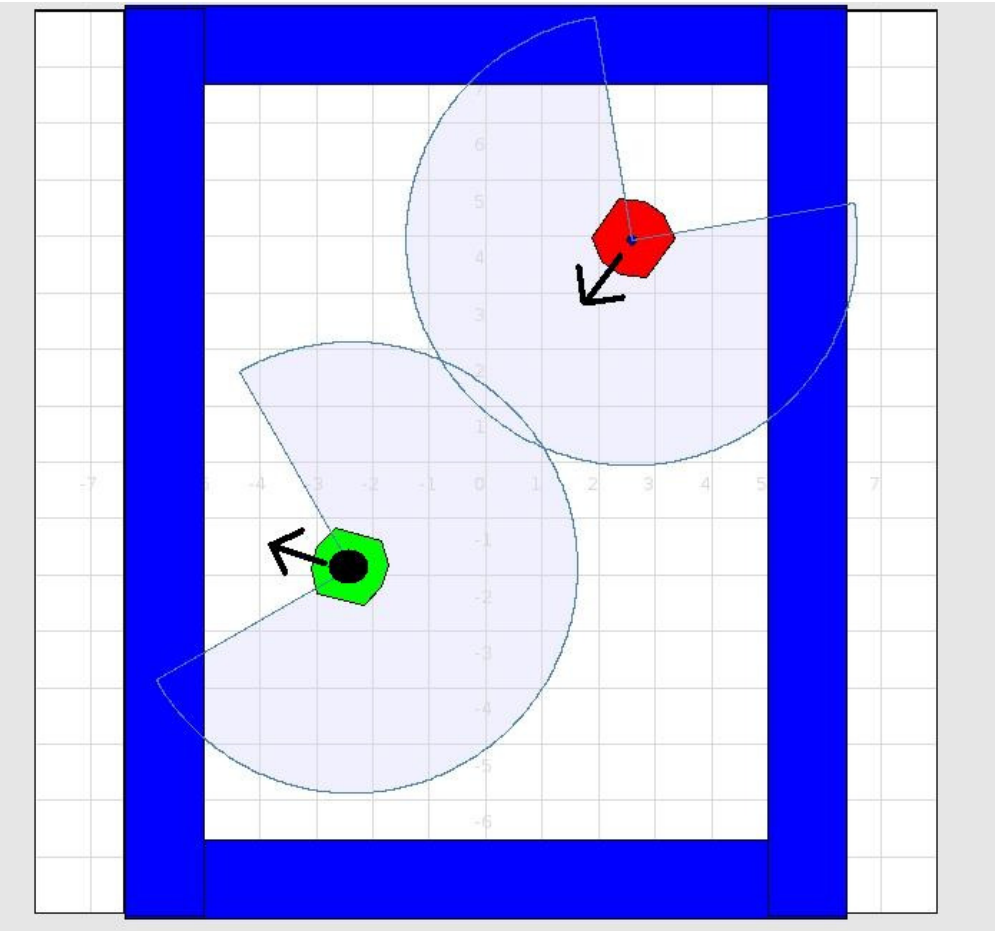


Figura 50 – Robôs predador e presa no ambiente.

A área azulada em volta dos robôs na figura 50 representa a área de detecção dos sensores ultrassom.

No sistema experimental estas áreas/linhas azuladas receberão a cor branca e a parte interna será preta para mais fácil detecção pelos sensores infravermelhos.

Inicialmente vêem-se o ambiente simulado com as paredes virtuais, o robô predador, em vermelho, e o robô presa, em verde e com um círculo preto em seu centro. As respectivas áreas cobertas pelos sensores, no robô predador, localizadas à frente, e no robô presa, na traseira, são representadas na figura 50, assim como as direções de locomoção, indicadas por setas.

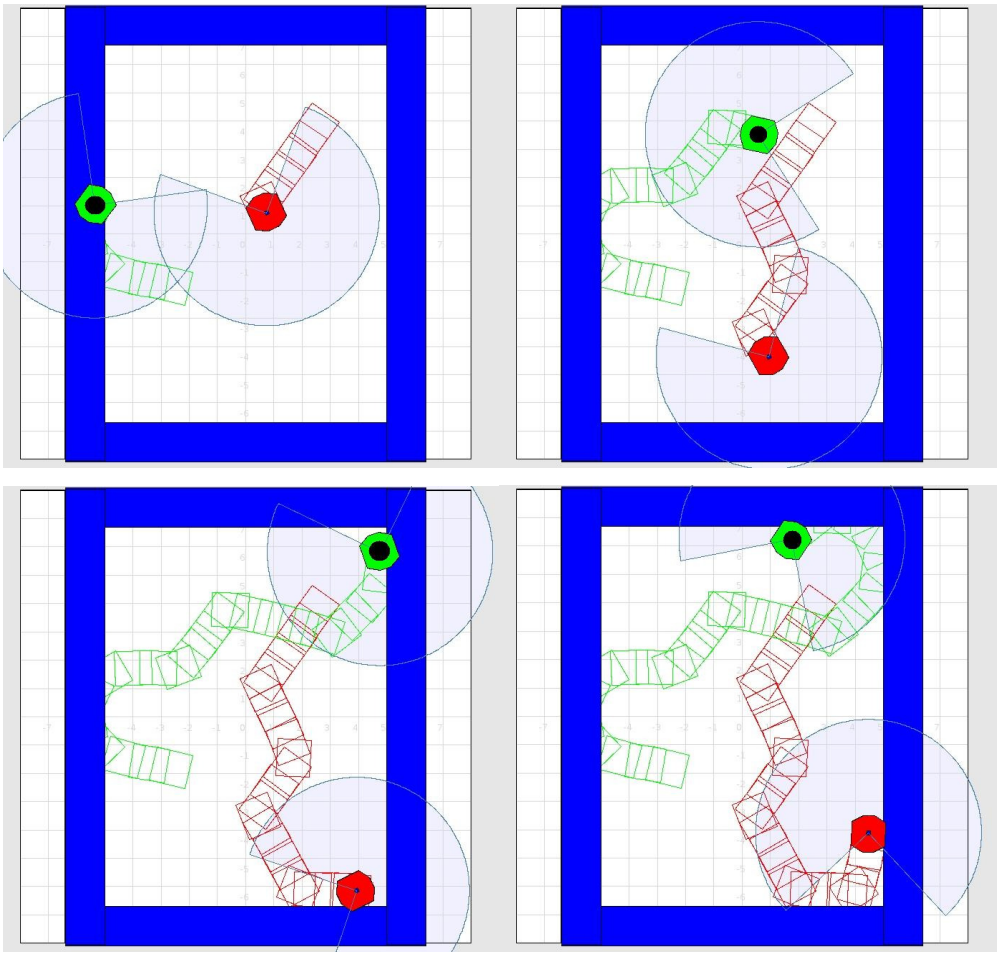


Figura 51 – Trajetória dos robôs predador e presa navegando pelo ambiente.

Iniciado o processo, vêem-se na figura 51 os robôs com o esquema motor EXPLORAR habilitado pelos respectivos arbitradores. O esquema perceptivo do predador inicia seu funcionamento, executando a navegação programada para andar durante um determinado tempo e alternar entre os lados de giro. Já no robô presa, o campo potencial repulsivo gerado pela faixa de desvio produz forças de repulsão que fazem com que o robô gire e se afaste da mesma. Nota-se que o robô presa entrou na faixa de desvio quase que de frente, o que fez com que os três esquemas perceptivos Desvia, Colisão e Batida contribuíssem consideravelmente para o sucesso do desvio.

Logo após, ainda na figura 51, é possível ver a trajetória executada pelos robôs executadas a partir dos respectivos esquemas perceptivos AndaGira, até que ambos se deparam com a faixa de desvio e próximos dos cantos da arena. O robô predador se aproxima com ângulo bem aberto, o que torna o desvio mais suave,

pois se tem tempo suficiente para executar a manobra. Já a presa entra mais uma vez com angulação acentuada, o que faz com que a manobra de desvio seja uma curva suave.

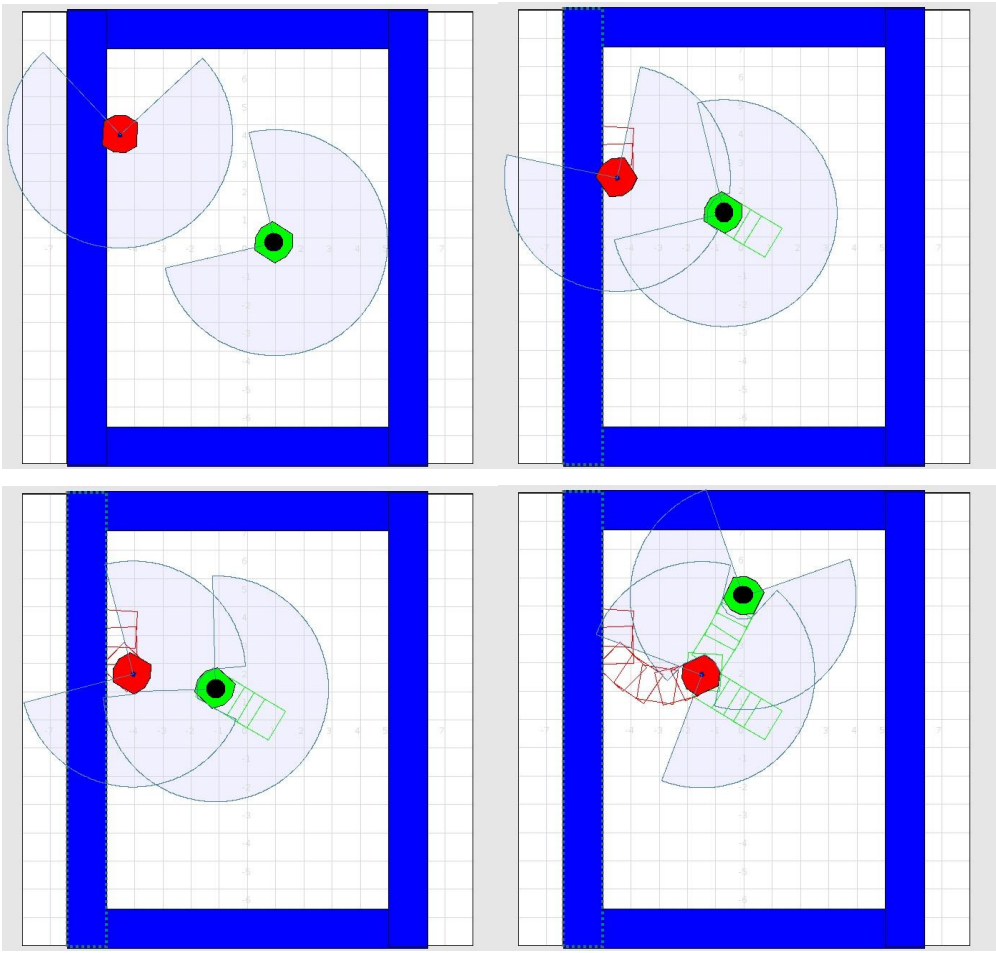


Figura 52 – Trajetória dos robôs presa desviando do robô predador.

Na figura 52, o robô presa está em rota de encontro com o robô predador. Assim que entra no alcance dos sensores do predador, este tem seu esquema motor CAÇAR ativado pelo arbitrador, e o campo potencial atrativo da presa exerce as forças necessárias para que direcione o predador diretamente para a presa. No momento seguinte a presa detecta o predador, o esquema motor FUGIR é habilitado, e o campo potencial repulsivo da presa gera as forças necessárias para realizar o desvio com sucesso.

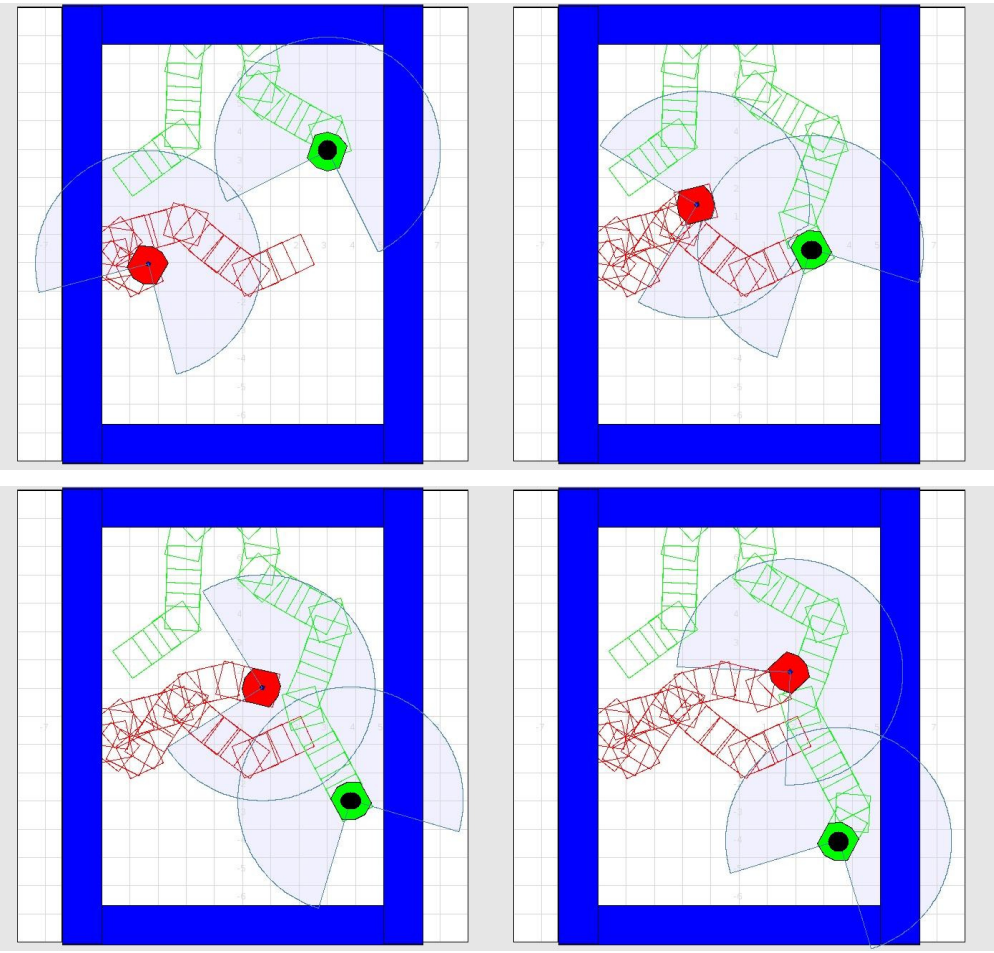


Figura 53 – Trajetória de fuga do robô presa.

Na figura 53, as trajetórias executadas pelos robôs resultam em uma rota na qual os robôs se identificam no mesmo instante, e a partir deste momento ambos têm seu respectivo esquema motor acionado.

A presa executa um leve desvio, enquanto que o robô predador é forçado a girar em um ângulo maior para poder perseguir a presa. Este giro faz com que o predador perca tempo e, devido à maior velocidade de presa, esta consegue sair do alcance dos sensores do predador, executando a fuga com sucesso.

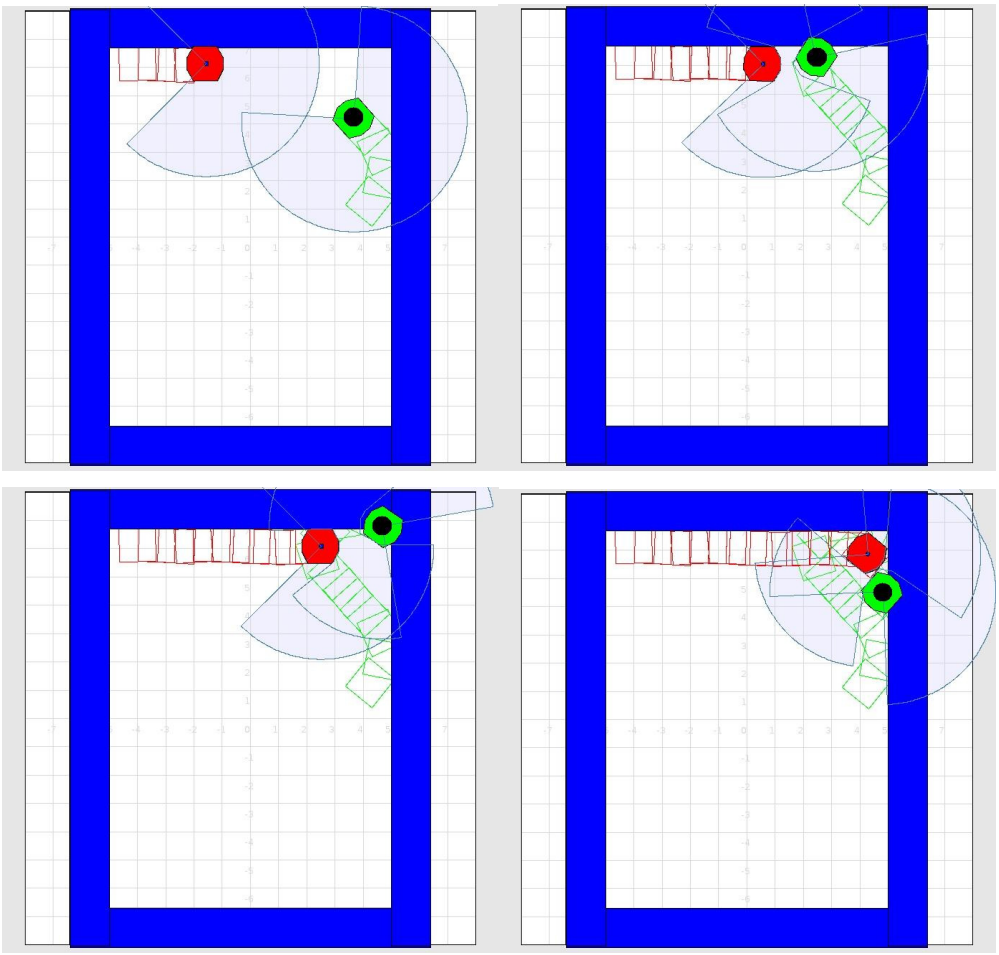


Figura 54 – Robô predador atinge robô presa.

Na figura 54, observa-se um exemplo de quando a presa é alcançada pelo predador. O predador mantém uma trajetória retilínea, pois quando seu esquema perceptivo tende a girá-lo para a esquerda, o esquema perceptivo de desvio corrige sua trajetória. Este fato faz parecer que ele obedece a uma rota constante.

O robô presa está localizado no canto superior direito da arena e com ângulo aberto. Quando detecta o predador com seu sensor esquerdo, ele reage para girar à direita. Esta ação coloca o robô presa em uma situação de risco, pois sobra pouco espaço para uma manobra evasiva sem sair da arena. Consequentemente, a perda de tempo gerada pelo desvio da faixa de perigo faz com que o robô predador consiga atingir o robô presa.

As simulações predador/presa realizadas são validadas experimentalmente com a utilização de robôs reais que foram utilizados como modelo para esta simulação. Os experimentos são descritos a seguir.

5 Validação Experimental

Os experimentos realizados têm como objetivo validar o controle baseado em comportamento por meio de robôs reais, situados em uma arena definida, e utilizando para isso a teoria predador / presa.

5.1. Robôs

Dois robôs identicamente construídos de dimensões 156mm x 116mm x 111mm são os objetos de estudo deste experimento. (Figura 55).

Cada robô é movido por um par de esteiras acionadas por motores de corrente contínua *Integy Matrix Pro Lathe 75T* e caixa de redução *Banebots* 36mm 16:1 responsáveis pela locomoção sendo um pra cada esteira, ou seja, trata-se de uma locomoção diferencial. Os robôs são acionados por uma bateria de lítio polímero.

Sensores ultrassom e infravermelho para detecção de outros robôs e de faixas de perigo, como já definido.

A fim de realizar todo processamento de controle, é utilizado um micro-controlador que é responsável pela leitura dos sensores e atuação sobre as rodas com base na programação embarcada.

O micro controlador utilizado foi o PIC 18F2550 da empresa *MicroChip,* pois apresenta baixo custo e características satisfatórias como: dez entradas analógicas, memória *flash* de 32k, memória RAM de 2048 bytes, *clock* de 48Mhz, duas saídas PWM e *buffer* para conexão USB (MICROCHIP, 2009)[18]

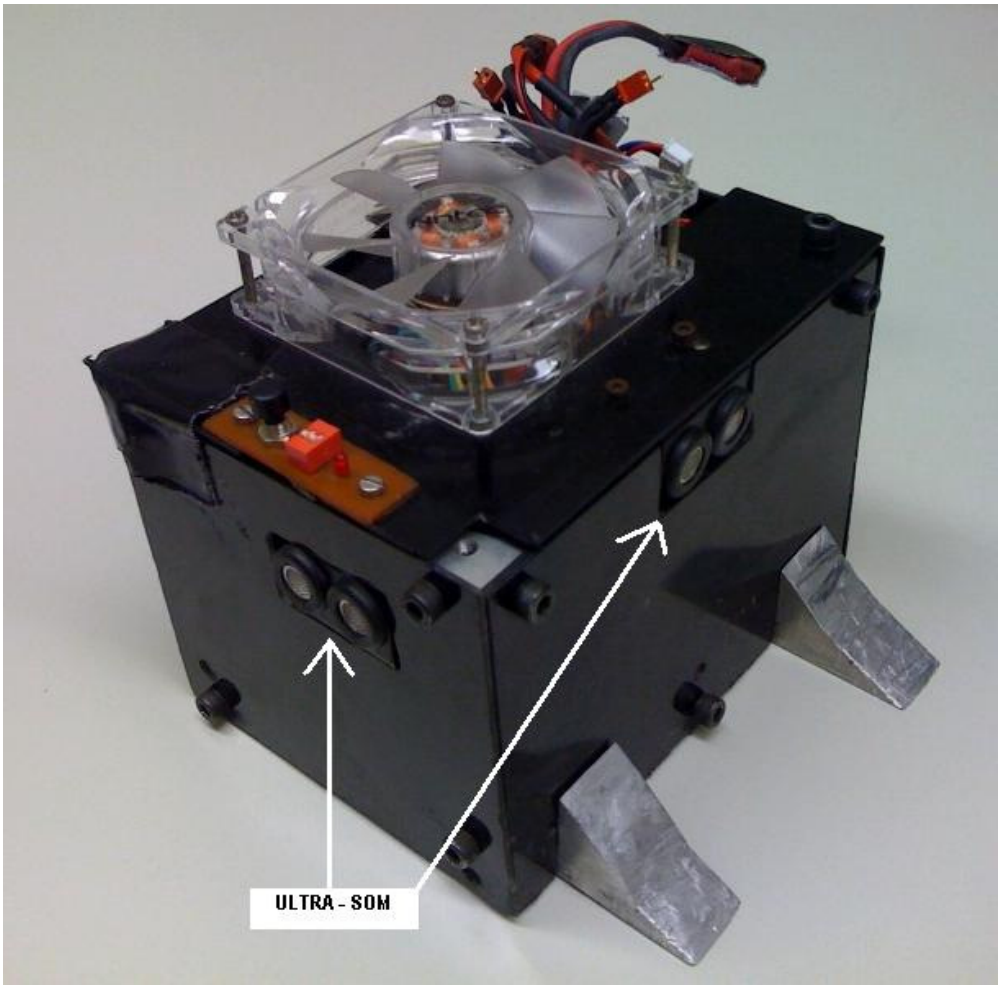


Figura 55 – Robô predador e respectivos sensores de ultrassom.

A única diferença entre os robôs é o sensoriamento de ultrassom. O robô predador possui um sensor ultrassom à frente e dois nas laterais esquerda e direita; já o robô presa possui um sensor ultrassom na traseira e os mesmos nas laterais.

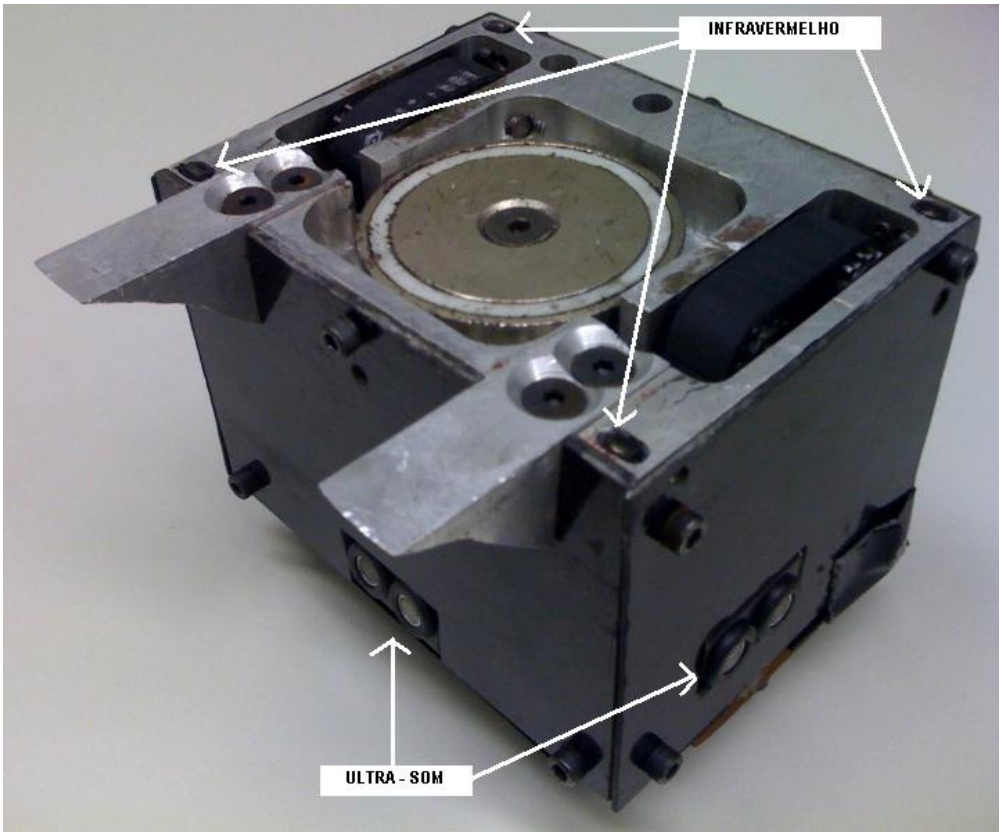


Figura 56 – Sensores infravermelhos localizados nas extremidades da base do robô.

Os sensores infravermelhos estão instalados na base dos robôs (Figura 56) e posicionados para baixo a fim de detectar as bordas da arena, delimitados por faixas brancas, análogas às faixas azuis da simulação.

Figura 57 – Robô predador, em verde, e robô presa, em azul, na arena.

A arena foi construída com uma chapa de aço 1020 de espessura 3mm, com a parte interior pintada de preto e a faixa externa em branco para facilitar a leitura por parte dos sensores infravermelhos.

5.2. Predador / Presa

Os robôs foram programados da mesma maneira que na simulação, ou seja, utilizou-se a programação baseada em comportamento com a arquitetura de esquemas motores aliada aos campos potenciais. A topologia também segue a mesma descrição com os esquemas motores EXPLORAR e CAÇAR para o robô predador, e EXPLORAR e FUGIR para o robô presa, assim como os esquemas perceptivos AndaGira, Desvia, Colisão, Ruído, Batida, Busca e Foge. (Figura 58).

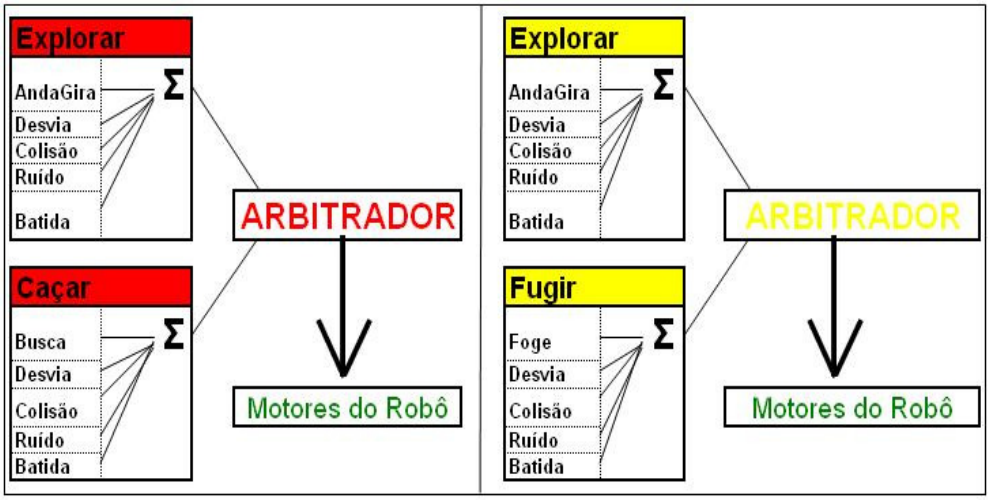


Figura 58 – Topologia dos robôs reais predador e presa programados com controle baseado em comportamento.

Assim como na simulação os valores atribuídos para as velocidades dos robôs são adimensionais e foram definidas para serem as mais altas possíveis para podermos comprovar o benefício do baixo custo de processamento do controle baseado em comportamento e para ser compatível com a simulação realizada.

Os valores para velocidades máximas dos robôs foram definidos entre (80 para frente) e -70 (para trás). Velocidades angulares variam entre -70 (esquerda) e 70 (direita). Os ganhos foram definidos de forma binária, assim como na simulação.

As figuras a seguir apresentam os comportamentos básicos de locomoção e desvio.

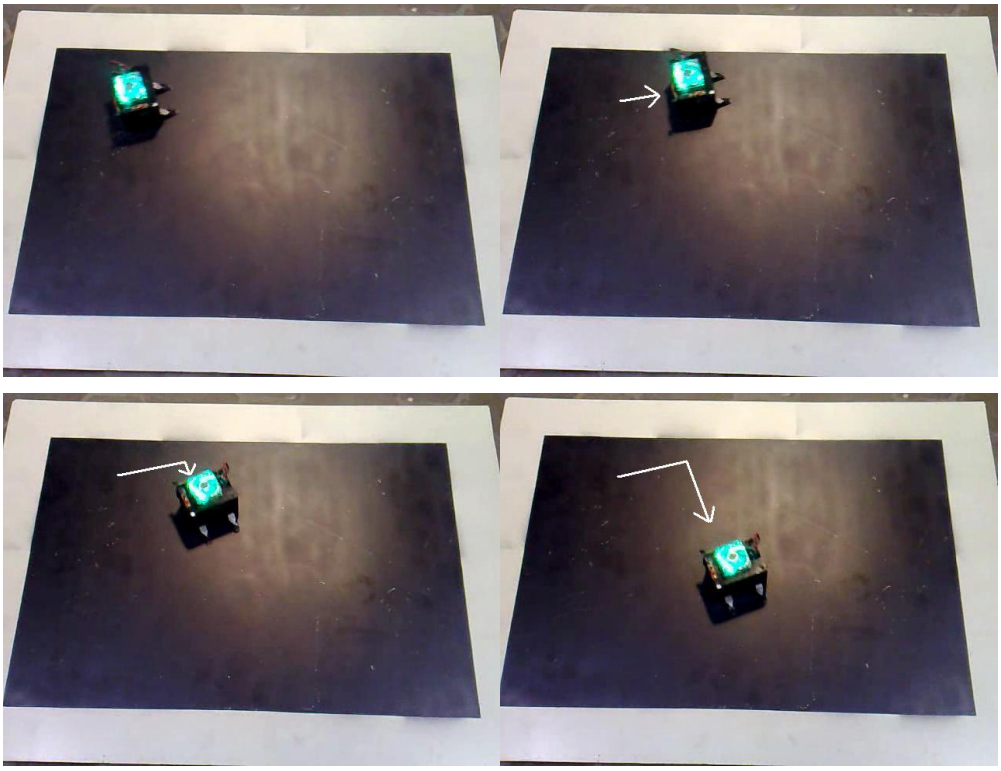


Figura 59 – Robô predador executando esquema motor EXPLORAR.

Na situação inicial da figura 59, o predador se encontra sozinho na arena apenas com o esquema motor EXPLORAR ativado, pois não há outro robô nas proximidades. Desta maneira, o esquema perceptivo AndaGira orienta o robô para executar a trajetória definida de exploração do ambiente, no qual o robô deve andar em linha reta durante 4s, e gira em sentidos alternados para melhor explorar o ambiente.

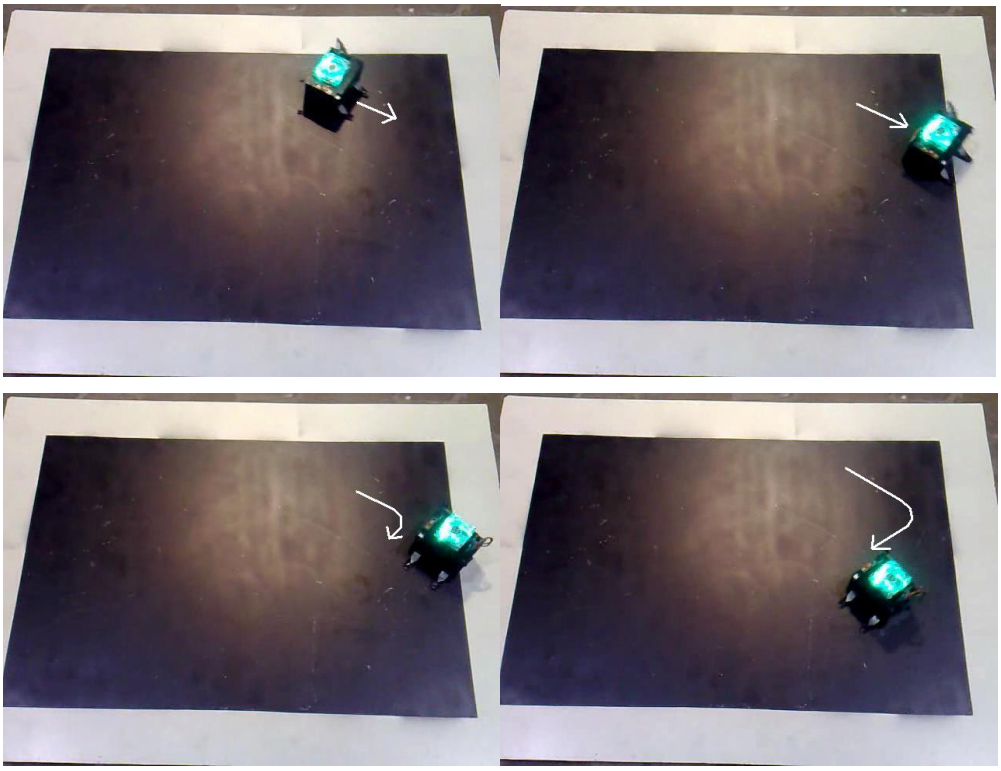


Figura 60 – Robô predador desviando da faixa de perigo.

Os esquemas perceptivos responsáveis pelo desvio das bordas do ambiente contribuem simultaneamente a partir da leitura dos sensores infravermelhos para o somatório vetorial final. Nota-se na figura 60, que o desvio da linha branca é efetuado com sucesso por parte do robô predador. Assim que o sensor infravermelho frontal esquerdo detecta a presença da faixa de desvio, o campo potencial repulsivo da borda contribui com alta magnitude para repelir o robô de volta para o interior da arena.

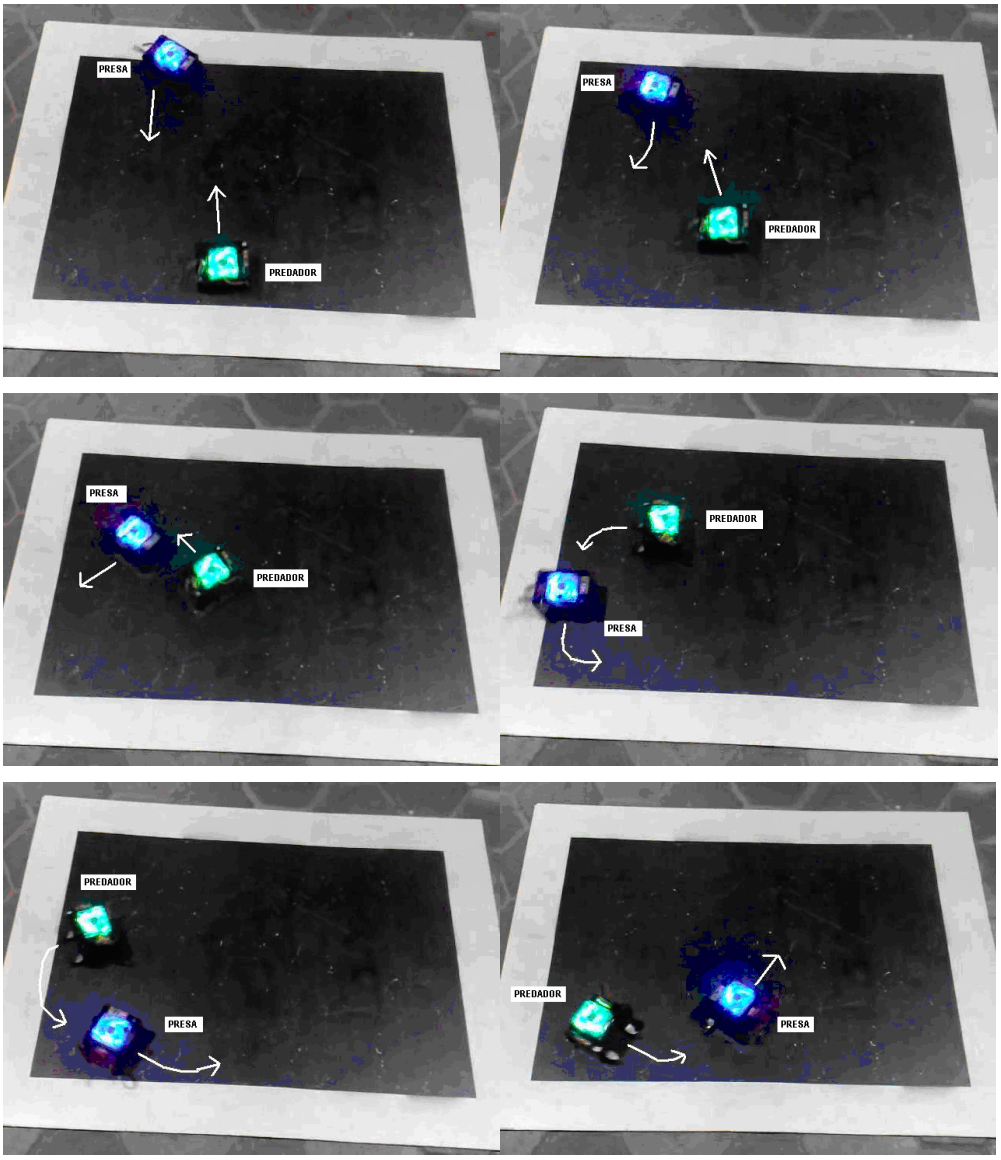


Figura 61 – Manobra de desvio do robô presa.

Na figura 61, os robôs entram em rota de encontro durante a exploração, e se detectam mutuamente no mesmo instante, contudo o campo potencial repulsivo criado pelo robô predador gera forças suficientes para fazer com que o robô presa consiga efetuar o desvio e evitar a colisão com o predador. Nestes e nos experimentos a seguir, a velocidade máxima do predador foi restringida para 80% da velocidade da presa. O robô predador por sua vez efetua a manobra de giro e perseguição do robô presa devido ao campo potencial atrativo criado pela presa. Os robôs seguem desviando das bordas das arenas.

Consegue-se observar que o robô presa consegue com sucesso evitar a colisão com o robô predador devido a não ter a necessidade de efetuar um grande giro sobre seu próprio eixo, e por ser um pouco mais veloz que o robô predador.

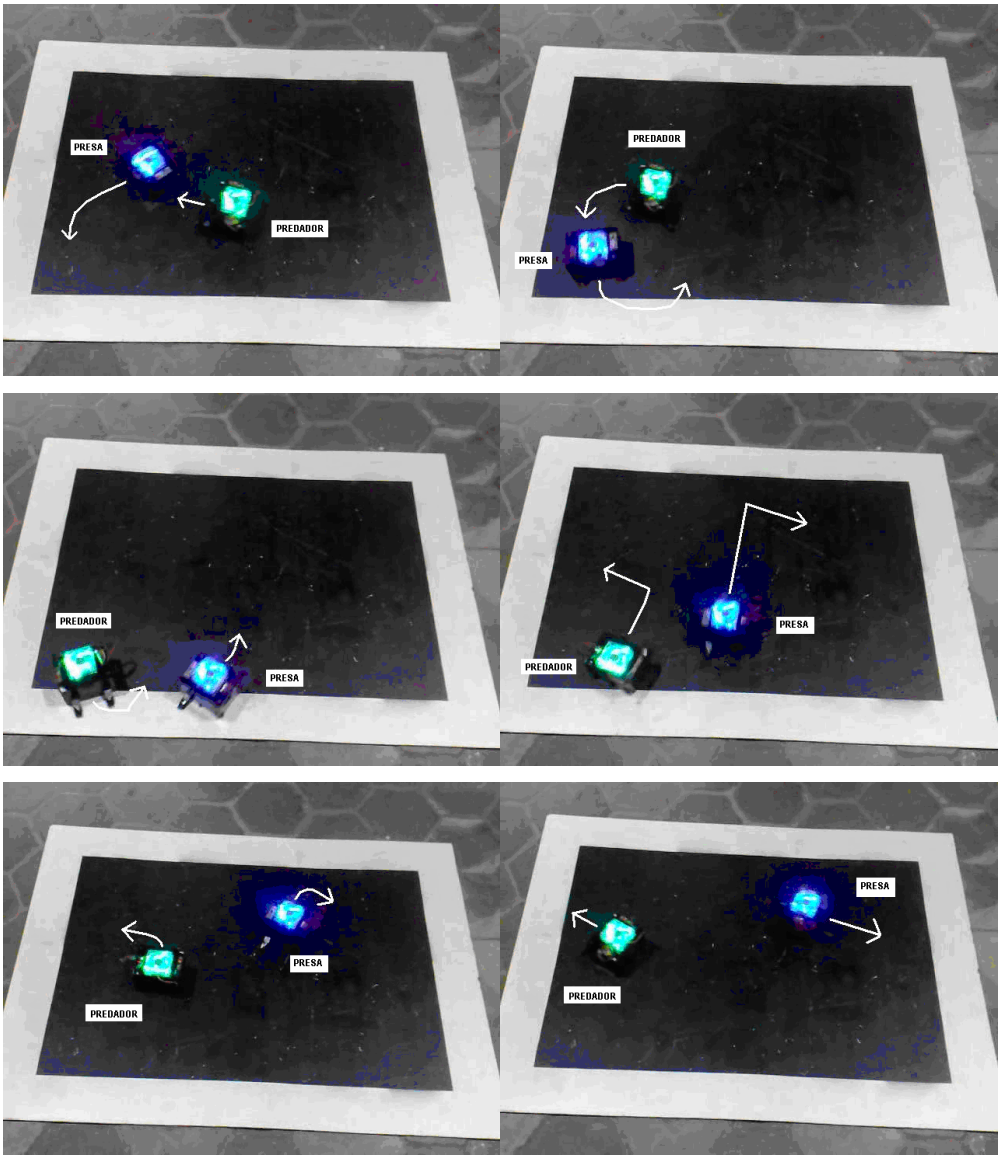


Figura 62 – Manobra de fuga do robô presa.

Na figura 62, a manobra de fuga executada pelo robô presa consiste em estabelecer uma trajetória capaz de sair do alcance dos sensores do predador. Têm-se os dois robôs muito próximos na primeira imagem, contudo o campo potencial repulsivo gerado pelo predador empurra o robô presa para a borda da arena em um ângulo aberto. Esta angulação favorece a rápida execução de desvio tanto da borda quanto do robô predador, visto que os campos potenciais de ambos estão se somando constantemente. O campo potencial que faz o predador voltar

para o interior da arena força este robô à executar uma curva para a esquerda atrapalhando a perseguição, porem ao voltar para o interior da arena a perseguição é retomada. À medida que o campo potencial do predador tende a empurrar a presa para fora da arena, o campo potencial da borda da arena contribui com alta magnitude para manter o robô dentro da área.

A partir deste somatório, nota-se que o robô presa consegue efetuar o desvio do predador com sucesso, e admite uma trajetória retilínea de fuga. Uma vez que o predador ainda está executando o desvio da faixa de perigo e a presa já assumiu um trajetória em linha reta, em um determinado momento o predador não consegue mais detectar a presa, o que causa em ambos o retorno às condições iniciais e encerra com sucesso a manobra de fuga por parte do robô presa.

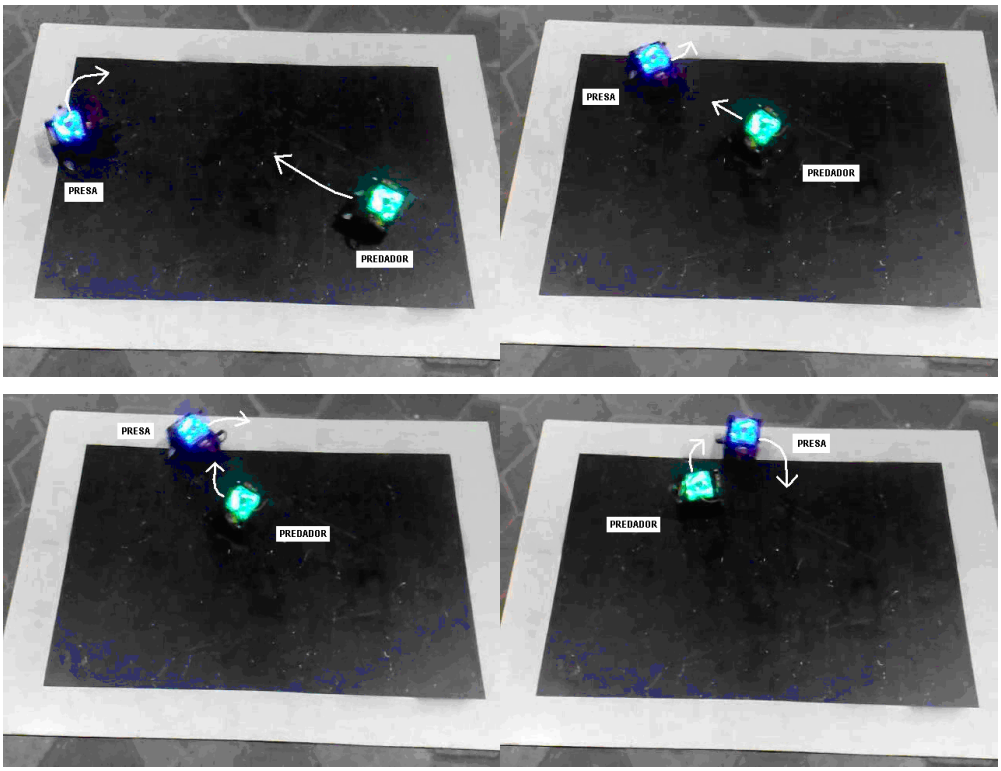


Figura 63 – Início da trajetória de captura da presa.

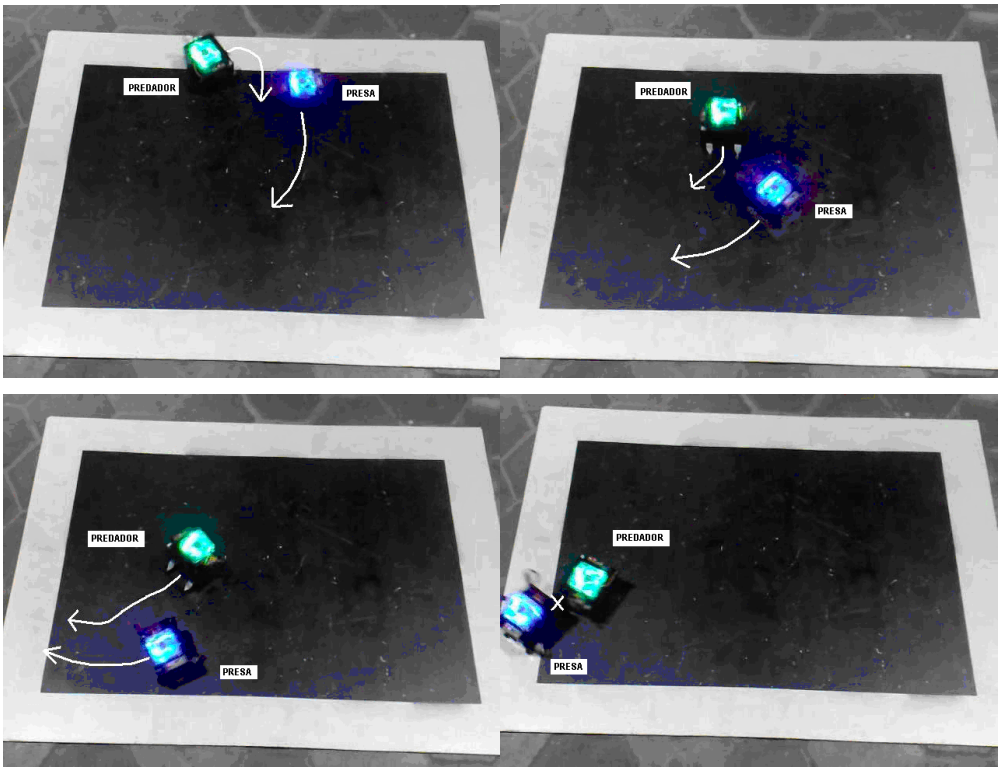


Figura 64 – Fim da trajetória de captura da presa.

No inicio do experimento, na figura 63, o robô predador detecta a presa e traça uma rota retilínea de interceptação.

A presa só identifica o robô predador quando este já se encontra muito próximo, é neste instante que o arbitrador aciona o esquema motor FUGIR. O campo potencial repulsivo repentino gerado pelo predador empurra a presa contra a borda da arena e com o somatório de forças do campo potencial da faixa de desvio o robô presa consegue, por pouco, evitar a colisão com o robô predador.

Contudo a perseguição se estende para o meio da arena, figura 64. As condições da trajetória impõem uma trajetória na qual o predador tende a encurralar a presa no canto inferior esquerdo da arena.

Quando o robô presa se aproxima do canto da arena e se depara com a faixa de perigo, ele inicia o desvio, mas a trajetória estabelecida pelo predador acarretou na colisão entre os robôs, o que caracteriza que o predador capturou a presa com sucesso.

Todos os experimentos realizados retrataram com certa fidelidade as simulações. Fatores como escorregamento das esteiras e pequenas interferências nos sensores ultrassom e infravermelhos contribuem para pequenas diferenças em

relação à simulação, contudo num contexto geral obteve-se a validação experimental de maneira satisfatória.

O acervo de equipamentos disponível até o momento destes experimentos não dispunha de um sistema de captação de movimento para que se pudessem executar comparações qualitativas. Entretanto, dezenas de repetições realizadas quantitativamente ajudaram a comprovar e validar o experimento junto às simulações.

6 Conclusões

O controle baseado em comportamento trouxe vantagens significantes no meio da robótica móvel autônoma, comprovadas pelas simulações realizadas.

A modularidade é uma característica de destaque. Comportamentos são agregados a um robô à medida que são criados, tornando a tarefa de programação mais ágil e eficaz.

Diferentes programadores podem trabalhar no mesmo projeto, criando módulos comportamentais diferentes, sem a necessidade de ter o conhecimento global do código e sem haver a necessidade de criar dependências ou refazer outras partes da programação. Inclusive programadores recém integrados em um projeto de controle podem desenvolver novos comportamentos sem conhecimento extenso do que já foi criado.

Esta afirmação pôde ser comprovada na simulação comparativa entre controle clássico e o baseado em comportamento, pois sempre que foi necessário adicionar uma rotina comportamental na arquitetura clássica, todo o código tinha que ser repensado e reescrito, enquanto que no controle baseado em comportamento apenas acrescentaram-se algumas linhas e suas respectivas referências ao somatório vetorial.

Caso algum módulo comportamental apresente falha, todos os outros continuam funcionando perfeitamente pois não há dependência entre eles, e com grandes chances de ainda assim alcançar o objetivo final, ou seja, aumenta-se consideravelmente a confiabilidade do sistema. A praticidade desta característica torna a programação mais intuitiva e dinâmica.

A velocidade do processamento proporcionada pela execução em paralelo dos comportamentos torna possível o máximo aproveitamento e otimização do poder computacional disponível, visto que os comportamentos, por definição, são módulos simplificados que não possuem dependências com outras partes do código fonte. Esta característica proporciona a criação de sistemas cada vez mais complexos.

A arquitetura de esquemas motores aliada à utilização da teoria dos campos potenciais mostrou ser eficiente quanto à suavidade nas respostas dos robôs, visto que contabiliza simultaneamente o somatório de forças geradas por todo o ambiente e conduz o robô de forma eficiente para o objetivo.

Contudo, no que diz respeito à exploração do ambiente à procura de objetos desejados, os robôs apresentaram altos tempos de exploração até encontrar o objetivo, pois em determinados casos passavam por lugares já explorados. Uma possível solução seria adotar um comportamento que pudesse armazenar ações e locais já trafegados, e com isso os robôs evitariam locais perigosos, com muitos obstáculos, ou onde tenham detectado presenças indesejadas recentemente, e dariam preferência a lugares com mais objetos de interesse.

O simulador Player/Stage, apesar de amplamente utilizado, requer um conhecimento avançado no sistema Linux para ser corretamente utilizado, o que demanda muito tempo de aprendizado.

As simulações e experimentos com a metodologia predador e presa comprovaram a eficiência da teoria do controle baseado em comportamento no que tange às aplicações reais desta teoria. Trajetórias qualitativamente similares foram obtidas em simulações e experimentos sob mesmas condições iniciais.

Outros focos do controle baseado em comportamento que podem ser desenvolvidos no futuro são:

- a aplicação dessas técnicas para uma grande quantidade de robôs atuarem em uma rede de comportamentos sociais cooperativa; e
- desenvolver o controle baseado em comportamento com aprendizado de curta e de longa duração, sem perder as características de modularidade e velocidade de processamento do controle.

A realização deste trabalho contribuiu para a observação de robôs operando com muitos comportamentos complexos atuando simultaneamente. Enquanto a referência bibliográfica mais completa (ARKIN, 1986) [1] exemplifica apenas cinco comportamentos primários, este trabalho utilizou dez comportamentos, que agregados geraram cinco comportamentos complexos.

Outra contribuição importante foi a validação experimental das simulações, que utilizaram as técnicas dos campos potenciais em robôs móveis autônomos, apresentando com isso todas as vantagens desta técnica.

Referências Bibliográficas

[1] Arkin, R. C., "Behavior-Based Robotics", MIT Press, Cambridge Massachusetts, 1998.

[2] Bonari, A., Mateucci, M., Resteli, M., "A Kinematics independent Dead Reckoning Sensor for indoor Mobile Robots", IEEE Robotics & Automation Magazine, 2003.

[3] Bonari, A., Mateucci, M., Resteli, M., "Automatic error detection and reduction for a odometric sensor based on two optical mice", IEEE Robotics & Automation Magazine, 2005.

[4] Sorenses, D. K., Smukala, V., Ovinis, M., Lee, S., "On-line optical flow feedback for mobile robot localization/navigation", IEEE Robotics & Automation Magazine, 2003.

[5] Kasper, M., Fricker, G., Puttkamer, E., "A based behavior architecture for teaching more then reactive behaviors to mobile robots", IEEE Robotics & Automation Magazine, 1999.

[6] Stenzel, R., "A behavior based architecture", IEEE Robotics & Automation Magazine, 2000.

[7] Emery, R., Balch, T., "Behavior based control of a non-holonomic robot in pushing tasks" IEEE Robotics & Automation Magazine, 2000.

[8] Huntsberger, T., Aghazarian, H., Baumgartner, E., Schenker, P. S., "Behavior based control systems for planetary autonomous robot outposts" IEEE Robotics & Automation Magazine, 2001.

[9] Li, W., Feng, X., "Behavior fusion for robot navigation in uncertain environments using fuzzy logic", IEEE Robotics & Automation Magazine, 1994.

[10] Fukayama, A., Ida, M., Katai, O., "Behavior based fuzzy control system for a mobile robot with environment recognition by sensory-motor coordination" IEEE Robotics & Automation Magazine, 1999.

[11] Taliansky, A., Shimki, N., "A behavior based navigation for an indoor mobile robot", IEEE Robotics & Automation Magazine, 2000.

[12] Suh, H., Lee, S., Kim, B., Yi, B., Oh, S., "Design and implementation of a behavior based control and learning architecture for mobile robots", IEEE Robotics & Automation Magazine, 2003.

[13] Li, H., Fu, Y., Xu, H., Ma, Y., "Avoiding static and dynamic objects in navigation", IEEE Robotics & Automation Magazine, 2006.

[14] Schenker, P.S., Huntsberger, T.L., Pirjanian, P., Baumgartner, E., Aghazarian, H., Trebi-Ollenu, A., Leger, P.C., Cheng, Y., Backes, P.G., Tunstel, E.W., "Robotic automation for space planetary surface exploration" MIT Press, 2003.

[15] Xu, L.W., Tso, S.K., "Sensor-based fuzzy reactive navigation of a mobile robot through local target switching", IEEE Robotics & Automation Magazine, 1999.

[16] Brooks, Rodney A. A robust Layered Control System for a Mobile Robot. IEEE Journal of Robotics and Automation RA-2, p. 14-23, 1986.

[17] Pro-Wave Electronic Corp, Disponível em: http://www.prowave.com .tw/english/item/ut.htm Acessado em 04/2009.

[18] MicroChip, Disponível em: http://www.microchip.com/wwwproducts/ Devices.aspx?dDocName=en010280 Acessado em 04/2009.

[19] FairChild Semiconductor, Disponível em: http://www.fairchildsemi. com/pf/QR/QRD1114.html Acessado em 05/2009.

[20] Jones, J., Flynn, A., Seiger, B., Mobile Robots: Inspiration to Implementation,second ed., A.K. Peters, 1998.

[21] Khatib, O., "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots." 1985 IEEE International Conference on Robotics and Automation, St. Louis, Missouri, March 25-28, 1990, pp.500-505

[22] Andrews, J. R. and Hogan, N., "Impedance Control as a Framework for Implementing Obstacle Avoidance in a Manipulator." Control of Manufacturing Processes and Robotic Systems, Eds. Hardt,D. E. and Book, W., ASME, Boston, 1983, pp. 243-251.

[23] Krogh, B. (1984). A generalized potential field approach to obstacle avoidance control. In: Proceedings of the SME Conf. Robotics Research, Bethlehem, PA.

[24] Medeiros, A. (1998). Introdução à robótica. In Anais do XVII Encontro Nacional de Automática, volume 1, pág. 56–65, Natal, RN, Brasil.

[25] Heinen, Farlei José. Robótica Autônoma: Integração entre Planificação e Comportamento Reativo; UNISINOS Editora, São Leopoldo, Novembro, 1999.

[26] Murphy, R.R.; "An Introduction to AI Robotics" - MIT Press pp400 1ªEdição, 2000.

[27] Pomerleau, D.; "No Hands Across America! (NHAA)" - Carnegie Mellon University Robotics Institute 1998.

[28] Lemonick, Michel. Dante Tours the Inferno. Time Magazine – Time Domestic/Science. Vol. 144, No. 7. August 15, 1994.

[29] Batavia, Parag; Pomerleau, Dean & Thorpe, Charles. Applying Advanced Learning Algorithms to ALVINN. CMU Technical Report CMU-RI-TR-96-31. Carnegie Mellon University. Pittsburgh. 1996.

[30] Paromtchik, I. E. & Laugier, C. Autonomous Parallel Parking of a Nonholonomic Vehicle. Procedings of the IEEE International Symposium on Intelligent Vehicles. pp. 13-18. September, 1996.

[31] Sheuer, A. & Laugier, C. Planning Sub-Optimal and Continuous-Curvature Paths for Car-Like Robots. IEEE- RSJ International Conference on Intelligent Robots and Systems. Victoria, British-Columbia, Canada. Oct. 1998.

[32] Russell, Stuart J.; Norvig, Peter (2003), Artificial Intelligence: A Modern Approach (2nd ed.), Upper Saddle River, NJ: Prentice Hall.

[33] Mataric, M. J. "Behavior-based control: Examples from navigation, learning and group behavior," J. Experimental Theoretical Artif. Intell, Special Issue: Software Architecture Phys. Agents, vol. 9, pp. 46–54, 1997.

[34] Nehmzow, U. , "Mobile Robotics: A Practical Introduction", Springer, London, 2000.

[35] Gat, E., "Integrating planning and reacting in a heterogeneous asynchronous architecture for mobile robots," in SIGART Bulletin 2, 1991, 70-74.

[36] Arkin, R., 1987, “Motor-Schema Based Navigation for a Mobile Robot: An Approach to Programming by Behavior”. IEEE International Conference on Robotics and Automation, Raleigh, NC, 264-271.

[37] Arbib, M.A., "Schema Theory", in the Encyclopedia of Artificial Intelligence, 2nd Edition, Editor Stuart Shapiro, 2:1427-1443, Wiley, 1992.

[38] Arkin, R. C. 1989b. "Motor Schema-Based Mobile Robot Navigation," International Journal of Robotics Research, Vol. 8, No.4, pp. 92-112.

[39] Arkin, R. C. 1992. "Behavior-Based Robot Navigation for Extended Domains," Adaptive Behavior, Vol. No.2, pp. 201-225.

[40] Cameron, J., MacKenzie, D., Ward, K., Arkin, R., and Book, W. 1993. "Reactive Control for Mobile Manipulation;' Processing of the International Conference on Robotics and Automation, Atlanta, GA, pp. 228-35.

[41] Khatib, O. “Real-time obstacle avoidance for manipulators and mobile robots”. In Proceedings of the IEEE Conference on Robotics and Automation, pages 500{505, 1985).

[42] Krogh, BH. “A Generalized Potential Field Approach to Obstacle Avoidance” Control Robotics International of SME (1984)

[43] Braitenberg, V. 1984. Vehicles: Experiments in Synthetic Psychology, MIT Press, Cambridge, MA.

[44] Benda, M, V. Jagannathan, and R. Dodhiawalla, “On optimal cooperation of knowledge sources”, Technical Report, Boeing Advanced Technology Center, Boeing Computer Services, Seattle, WA, 1986.

[45] Goldberg, Dani and Mataric, Maja J (2001) Design and Evaluation of Robust Behavior-Based Controllers for Distributed Multi-Robot Collection Tasks. Robot Teams: From Diversity to Polymorphism.

[46] Stone, Peter (2007) “Intelligent Autonomous Robotics: A Robot Soccer Case Study” Synthesis Lectures on Artificial Intelligence andMachine Learning. ISBN: 1598291262

[47] Mataric, M. J., 1997, "Behavior Based Robotics as a Tool for Synthesis of Articial Behavior and Analysis of Natural Behavior" Trends in Cognitive Science, Vol.2 N.3 March 1998, 82-87

[48] Brooks, Rodney A. "Cambrian Intelligence: The Early History of the New AI". MIT Press, Cambridge,Massachusetts, 1999

[49] Maes, P. & Brooks, R. A. (1990), Learning to Coordinate Behaviours, in "Proceedings, AAAI-91", Boston, MA, pp. 796-802.

[50] Connell, J. H. (1990), Minimalist Mobile Robotics: A Colony Architecture for an Artificial Creature, Academic Press.

[51] Rosenblatt, J., DAMN: a distributed architecture for mobile navigation. J. Exp. Theor. Artif. Intell. 9(2-3): 339-360 (1997)

[52] ROSENSCHEIN, S.J. & L.P. KAELBLING, The synthesis of digital machines with provable epistemic properties. In J.Y. Halpern (ed.), Theoretical aspects of reasoning about knowledge: Proc. Fifth Conference, p. 83-97. San Francisco (Cal.) Morgan Kaufmann, 1986.

[53] Zeltzer, D. and Johnson, M. B., Motor Planning: Am architecture for specifying and controlling the behaviour of virtual actors. Journal of Visualization and Computer Animation, 2:74-80, 1991.

[54] Firby, R. J. and Earl, C. Combined Execution and Monitoring for Control of Autonomous Agents. 1988-1995

[55] iRobot Robotics, Disponível em: http://store.irobot.com/home/index.jsp Acessado em 06/2009.

[56] NASA Rovers, Disponível em: http://marsrover.nasa.gov/home/index.html Acessado em 06/2009.

[57] Boston Dynamics, Disponível em: http://www.bostondynamics.com/robot_bigdog.html Acessado em 12/2009.

[58] Wolf, Denis F., Disponível em: www.icmc.usp.br/~denis/files/Player_man_v20.pdf Acessado em 12/2009.

Apêndice I

Arquivos com código fonte de cada simulação e experimento que acompanham este trabalho são especificados a seguir:

motorschema.cpp > Código fonte da simulação do robô principal com controle baseado em comportamento.

inimigo.cpp > Código fonte da simulação do robô inimigo/predador com controle baseado em comportamento.

motorcomp-bbr.cpp > Código fonte da simulação do robô principal com controle baseado em comportamento programado com comportamentos simplificados para comparação.

inimigocomp-bbr.cpp > Código fonte da simulação do robô inimigo/predador com controle baseado em comportamento programado com comportamentos simplificados para comparação.

motorcomp-classic.cpp > Código fonte da simulação do robô principal com programação de controle clássica para comparação.

inimigocomp-bbr.cpp > Código fonte da simulação do robô inimigo/predador com programação de controle clássica para comparação.

predador.cpp > Código fonte da simulação do robô predador.

presa.cpp > Código fonte da simulação do robô presa.

tese1-laser.cfg > Definições da simulação principal executada no software player/stage.

tese1-laser.world > Definições do ambiente e dos robôs na simulação principal executada no software player/stage.

tese1-comp.cfg > Definições da simulação simplificada executada no software player/stage para comparação entre diferentes tipos de programação.

tese1-comp.world > Definições do ambiente e dos robôs na simulação simplificada executada no software player/stage para comparação entre diferentes tipos de programação.

tese-sumo.cfg > Definições da simulação dos robôs predador e presa executada no software player/stage.

tese-sumo.world > Definições do ambiente e dos robôs na simulação dos robôs predador e presa executada no software player/stage.

santerio.c > Código fonte do experimento do robô predador.

santerio-pres.c > Código fonte do experimento do robô presa.

Apêndice II

O arbitrador do controle baseado em comportamento funciona de maneira cíclica. Realiza o sequenciamento dos esquemas motores à medida que os sensores fornecem informações do ambiente. A figura 37 mostra a sequência de execução dos comportamentos complexos. Este sequenciamento é feito utilizando o operando "*if*" da linguagem de programação C++. Trabalha-se com todos os esquemas perceptivos com pesos binários, ou seja, contribuem ou não para a resposta do robô de acordo com o esquema motor ativo no momento.

Os valores de saída são parâmetros internos do simulador e empiricamente verificou-se que equivalem a:

Velocidade máxima, de valor 1, equivale a cerca de 0,5 m/s.

Angulação, que varia entre 5 e -5, equivale a variação entre 180º e -180º.

O esquema motor EXPLORAR, por exemplo, não necessita que haja colaboração do esquema perceptivo **entrega**, pois no momento a ordem é explorar o ambiente à procura do objeto de interesse. O arbitrador então atribui peso zero ao esquema perceptivo **entrega** e a outros esquemas perceptivos que não devem contribuir para a saída final. Atribui o valor um aos esquemas importantes para execução da tarefa. Um pseudocódigo para o comportamento EXPLORAR seria:

```
If  não detecta objeto   {      // Comportamento EXPLORAR Ativo
  peso andagira = 1;            // Esquema Perceptivo Ativado
  peso foge = 1;                // Esquema Perceptivo Ativado
  peso desvia = 1;              // Esquema Perceptivo Ativado
  peso colisão = 1;             // Esquema Perceptivo Ativado
  peso ruido = 1;               // Esquema Perceptivo Ativado
  peso batida = 1;              // Esquema Perceptivo Ativado
  peso busca = 0;               // Esquema Perceptivo Desativado
  peso abrir garra = 0;         // Esquema Perceptivo Desativado
  peso fechar garra = 0;        // Esquema Perceptivo Desativado
  peso entrega = 0;       }     // Esquema Perceptivo Desativado
```

Pesos diferentes podem ser usados quando se necessita de um controle mais fino ou atribuir mais importância a determinado esquema. Podem-se criar funções lineares para determinar a variação dos pesos, contudo neste trabalho verificou-se que a aplicação de pesos binários é satisfatória para executar o controle dos robôs.

O somatório resultante, neste momento, é calculado somando-se diretamente as contribuições de ângulos e velocidades multiplicados pelos respectivos pesos, ou seja, neste caso se tem:

```
somatorio-> velocidade =
                    andagira->velocidade * andagira->peso
                  + busca->velocidade * busca->peso
                  + entrega->velocidade * entrega->peso
                  + foge->velocidade * foge->peso
                  + desvia->velocidade * desvia->peso
                  + colisao->velocidade * colisao->peso
                  + ruido->velocidade * ruido->peso
                  + fechar_garra->velocidade * fechar_garra->peso
                  + abrir_garra->velocidade * abrir_garra->peso
                  + batida->velocidade * batida->peso
                                                                    ;
somatorio-> angulo =
                    andagira->angulo * andagira->peso
                  + busca->angulo * busca->peso
                  + entrega->angulo * entrega->peso
                  + foge->angulo * foge->peso
                  + desvia->angulo * desvia->peso
                  + colisao->angulo * colisao->peso
                  + ruido->angulo * ruido->peso
                  + fechar_garra->angulo * fechar_garra->peso
                  + abrir_garra->angulo * abrir_garra->peso
                  + batida->angulo * batida->peso  ;
```

Os valores instantâneos para cada parâmetro neste exemplo são:

andagira->velocidade = 0,6 e andagira->ângulo = 2

busca->velocidade = 0 e busca->ângulo = 0

entrega->velocidade = 0 e entrega->ângulo = 0

foge->velocidade = -0,2 e foge->ângulo = -1

desvia->velocidade = 0,3 e desvia->ângulo = -0,4

colisao->velocidade = 0 e colisao->ângulo = 0

ruido->velocidade = 0,02 e ruido->ângulo = 0,02

fechar_garra->velocidade = 0 e fechar_garra->ângulo = 0

abrir_garra->velocidade = 0 e abrir_garra->ângulo = 0

batida->velocidade = 0 e batida->ângulo = 0

O somatório resultante é:

somatorio-> velocidade = (0,6 * 1) + (0 * 0) + (0 * 0) + (-0,2 * 1) + (0,3 * 1) + (0 * 1) + (0,02 * 1) + (0 * 0) + (0 * 0) + (0 * 1) = 0,72 = 0,36m/s

somatorio-> angulo = (2 * 1) + (0 * 0) + (0 * 0) + (-1 * 1) + (-0,4 * 1) + (0 * 1) + (0,02 * 1) + (0 * 0) + (0 * 0) + (0 * 1) = 0,62 = 22,32º

A fim de evitar que o resultado do somatório exceda os limites mecânicos para velocidade e angulação criou-se um equacionamento que estabelece limites máximos para estes parâmetros:

somatorio->velocidade=(somatorio->velocidade > 1) ? 1 : somatorio->velocidade;

somatorio->velocidade=(somatorio->velocidade < - 0,7) ? – 0,7 : somatorio->velocidade;

somatorio-> angulo = (somatorio-> angulo > 5)? 5 : somatorio-> angulo;

somatorio-> angulo = (somatorio-> angulo < -5)? -5 : somatorio-> angulo;

Desta forma garante-se que a velocidade máxima em frente dos motores, equivalente ao valor de saída 1 do simulador, a velocidade máximo para traz equivale ao valor -0,7 de saída, o ângulo máximo e mínimo varia entre 5 e -5.

Este exemplo não saturou os limites, portanto a saída final manteve-se:

somatorio-> velocidade = 0,36m/s

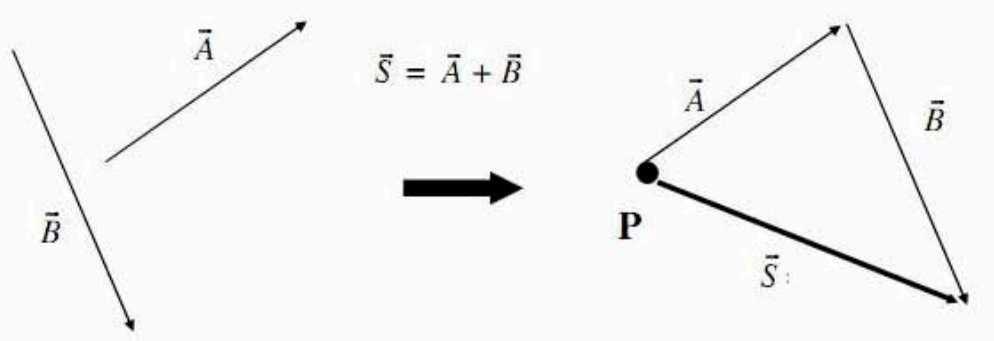
somatorio-> angulo = 22,32º

O software de controle real executa automaticamente uma função interna que gera a saída real em volts, para cada motor do robô.

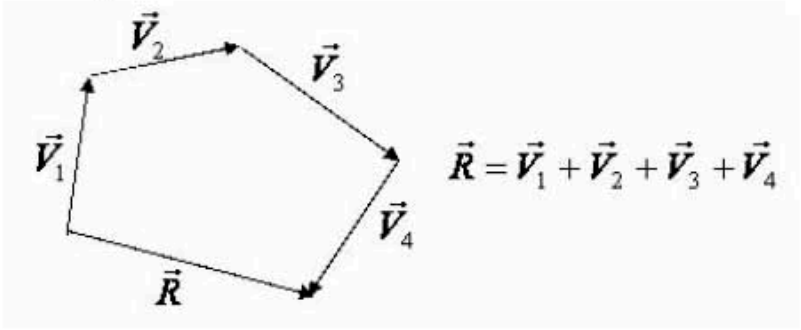
O esquema motor SOLTAR é um comportamento que não precisa de contribuições de outros esquemas perceptivos, a não ser o esquema **foge**. Neste esquema, o robô deve se manter imóvel para liberar o objeto de forma correta. Qualquer outra contribuição causaria instabilidade no processo.

Este sequenciamento pode ser feito de maneira linear através de uma função entre os pesos dos esquemas perceptivos, contudo a utilização de pesos binários mostrou-se satisfatória para as simulações executadas.

O somatório das contribuições dos esquemas perceptivos, ou seja, a sobreposição dos campos potenciais gerados no ambiente é realizada na forma de soma vetorial.



Cada esquema perceptivo gera um vetor resultante com magnitude e direção do robô. A soma vetorial destes vetores gera o vetor resultante que indica o sentido e a magnitude da velocidade do robô no instante calculado.

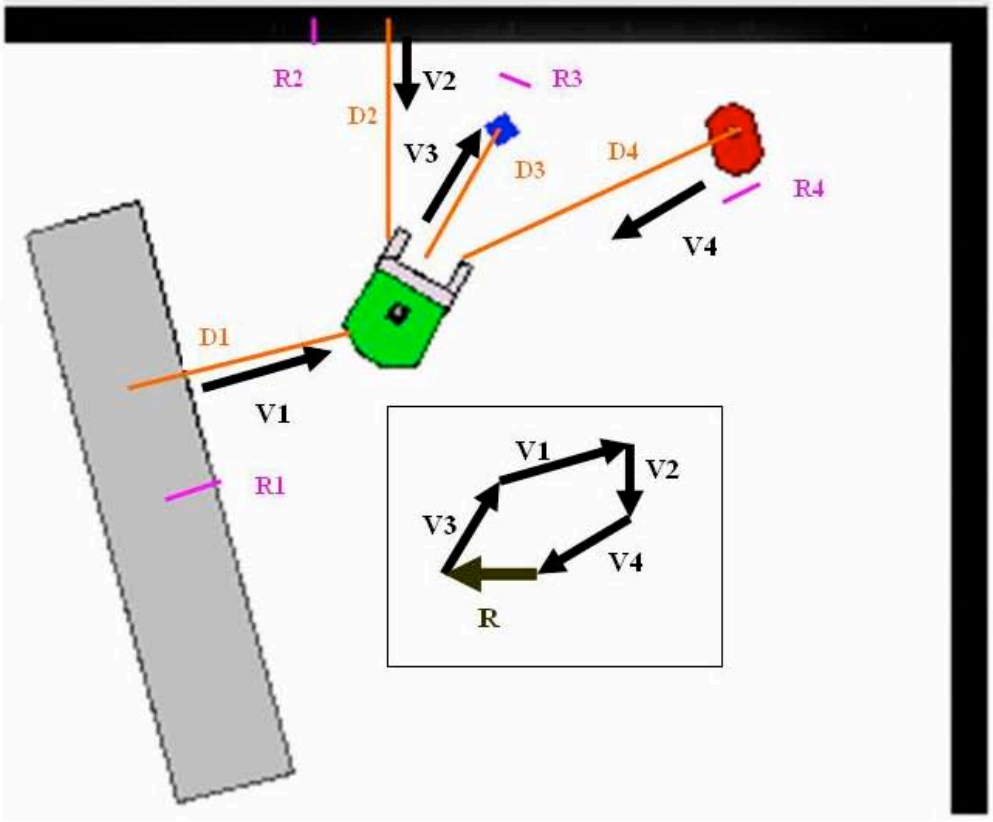


O exemplo a seguir apresenta o somatório das componentes dos campos potenciais atrativo, repulsivo e perpendicular gerados no ambiente simulado.

Os vetores determinam a magnitude da velocidade, obtida pelo seu módulo e com unidade inerente ao software, convertida em m/s e posteriormente em tesão elétrica para os motores. Os vetores também indicam direção e sentido do movimento em valores internos e convertidos para graus pela função de conversão do software.

Os campos potenciais são criados por seus respectivos esquemas perceptivos e suas características, entretanto, alguns esquemas perceptivos geram apenas campos vetoriais para locomoção como o andagira, ruído, fechar_garra e abrir_garra. Nestes esquemas os campos vetoriais são pré-definidos e somam-se aos campos potenciais em cada instante de execução. O software recebe os vetores dos campos potenciais e dos campos vetoriais e os interpreta da mesma forma, como vetores que possuem magnitude e direção instantânea do robô.

O obstáculo cinza gera um campo potencial repulsivo representado pelo vetor V1. A parede preta gera um campo perpendicular V2. O objeto azul gera um campo atrativo de componente instantânea V3, o robô predador gera um campo de repulsão com magnitude e direção contida no vetor V4 e as distâncias destes objetos até o robô de coleta são respectivamente D1, D2, D3 e D4.



Atribuiu-se para este exemplo os valores numéricos para as distancias dos centros radiais dos elementos até o robô e seus respectivos raios a serem descontados nas equações: D1 = 120mm; D2 = 110mm; D3 = 70mm; D4 = 125mm; R1 = 20mm; R2 = 10mm; R3 = 10mm; R4 = 15mm;

As áreas onde as funções atuam dependem do alcance dos sensores tem os valores: S1 = 220mm ; S2 = 160mm ; S3 = 510mm; S4 = 315mm;

Os ganhos das funções são a representação adimensional dos pesos utilizados pelo somatório vetorial e são definidos de forma binária com valor 1, quando o campo potencial está ativo e como 0, quando não contribui para o somatório portanto, neste exemplo os ganhos estão definidos como: G1 = G2 = G3 = G4 = 1.

O software de simulação foi desenvolvido utilizando valores em cm para que a saída esteja compreendida entre os valores adimensionais -1 e 1 para velocidades e -5 e 5 para angulação. Posteriormente estes valores são convertidos para as velocidades em m/s e graus e estes são transformados em tensões elétricas que variam entre -14,8v e +14,8v para acionamento dos motores.

A direção de cada vetor é dada conforme a figura, e o cálculo dos módulos é feito de acordo com a função relacionada a cada campo potencial utilizado. Os módulos dos vetores V1, V2 e V4 são calculados a partir da equação que exerce uma força de repulsão proporcional ao quadrado da distância entre o objeto e o robô de coleta, respectivamente D1, D2 e D4, portanto:

$$|V1| = \frac{S1 - R1}{(D1 - R1)^2} \cdot G1 = \frac{20 - 2}{(12 - 2)^2} \cdot 1 = 0{,}2$$

$$|V2| = \frac{S2 - R2}{(D2 - R2)^2} \cdot G2 = \frac{16 - 1}{(11 - 1)^2} \cdot 1 = 0{,}15$$

$$|V4| = \frac{S4 - R4}{(D4 - R4)^2} \cdot G4 = \frac{31{,}5 - 1{,}5}{(12{,}5 - 1{,}5)^2} \cdot 1 = 0{,}25$$

O módulo do vetor V3 é calculado utilizando a função do campo potencial atrativo que exerce força de atração proporcional ao quadrado da distância D3.

$$|V3| = \frac{(D3 - R3)^2}{S3 - R3} G3 = \frac{(7 - 1)^2}{51 - 1} \cdot 1 = 0{,}72$$

O vetor resultante é calculado como no exemplo anterior, portanto temos: somatorio-> velocidade = (0,2 * 1) + (-0,15 * 1) + (0,72 * 1) + (-0,25 * 1) = 0,52 = 0,26m/s

Os ângulos, neste exemplo, são calculados pelo software com as mesmas equações da velocidade portanto para ângulo de V1 = 10º; ângulo de V2 = -90º; ângulo de V3 = 60º; ângulo de V4 = -160º.

somatorio-> angulo = (0,27 * 1) + (-2,5 * 1) + (1,66 * 1) + (-4,44 * 1) = -5 = -180º

A sobreposição dos campos potenciais gera o campo potencial resultante que determina a trajetória do robô no ambiente a cada instante. O momento destacado demonstra que no instante analisado o campo potencial repulsivo gerado pelo robô predador contribui com maior relevância para a trajetória do robô.

O simulador calcula as saídas conforme apresentado, contudo, o cálculo geral do vetor resultante R é dado pelo produto dos módulos dos vetores V1, V2, V3 e V4 por seus respectivos vetores unitários na direção do campo, v1, v2, v3 e v4.

$$R = (|V1| \cdot v1) + (|V2| \cdot v2) + (|V3| \cdot v3) + (|V4| \cdot v4)$$

$$R = \left(\frac{S1 - R1}{(D1 - R1)^2} \cdot G1 \cdot v1 \right) + \left(\frac{S2 - R2}{(D2 - R2)^2} \cdot G2 \cdot v2 \right) + \left(\frac{(D3 - R3)^2}{S3 - R3} G3 \cdot v3 \right) + \left(\frac{S4 - R4}{(D4 - R4)^2} \cdot G4 \cdot v4 \right)$$

Os somatórios são executados a todo instante e constantemente atualizados para todos os esquemas motores.

Primeiramente o código fonte resumido e explicativo para entendimento do conceito e logo em seguida código completo da simulação completa.

```
//  Código Fonte Resumido Programação Baseada em Comportamento
Utilizando Arquitetura de Esquemas Motores

//  Fabiano Santério - 2009

// Inicializando o robo e sensores

////////// Inicialização das Flags Dos Comportamentos
////////// Funcionam para a tomada de decisão do Arbitrador

int aquisitar_flag = 0;
int pegar_flag = 0;
int entregar_flag = 0;
int soltar_flag = 0;
```

```
////////////////// Definindo Estrutura Básica Do Vetor Resposta

struct resultante {

      float velocidade;       // Magnitude Vetor Velocidade
      float angulo;           // Ângulo
      float peso;             // Pesos dos Comportamentos
};

/////  Definindo COMPORTAMENTOS PRIMITIVOS (Esquemas Perceptivos)

// ANDA E GIRA -> Anda em linha reta durante um determinado tempo
e gira em seu próprio eixo

// BUSCA -> Busca Objeto AZUL id = 1
// Campo Potencial Atrativo

// ENTREGA -> Procura Local de Entrega (Objeto amarelo) id = 4
// Campo Potencial Atrativo

// FOGE - > Foge de objeto vermelho   id = 0
// Campo Potencial Repulsivo

//  DESVIA -> Desvia de Obstáculos Estáticos Utilizando Sonar
//  Campo Potencial Repulsivo

//  COLISAO -> Evita Colisão
//  Campo Potencial Repulsivo

// RUIDO -> Gera Ruído Aleatório para evitar situações singulares

// FECHAR_garra -> Aciona a Garra para pegar objeto.

// ABRIR_garra -> Aciona a garra para soltar objeto

// BATIDA -> Executa manobra após detectar colisão
//  Campo Potencial Repulsivo

////////////////////  FUNÇÃO PRINCIPAL     ////////////////

int main(void)
{
      robot.Read();     //Faz a primeira varredura nos sensores

   // Chamando os esquemas perceptivos

// Anda durante um tempo e gira em torno do próprio eixo
// Quando detecta objetivo por cor , vai em sua direção
// Quando detecta local de entrega por cor, vai em sua direção
// Quando detecta inimigo por cor, desvia
// Desvia de obstáculo usando Sonar
// Evita colisões
// Ruído Aleatório
// Fechar Garra
// Abrir Garra
// Batida

/////////////// Definindo Comportamentos  ////////////////////////

//// Arbitrador
```

```
      if (soltar_flag == 1)
      std::cout << "Comportamento Ativo ->  !! SOLTAR !! \n"
      else if (entregar_flag == 1)
      std::cout << "Comportamento Ativo ->  !! ENTREGAR !!\n"
      else if (pegar_flag == 1)
      std::cout << "Comportamento Ativo ->  !! PEGAR !!\n"
      else if (aquisitar_flag == 1)
      std::cout << "Comportamento Ativo ->  !! AQUISITAR !!\n"
      else
      std::cout << "Comportamento Ativo ->  !! EXPLORAR !!\n"

// Definindo Limites de Velocidade e Taxa de Rotação
// Atua nos motores do robô
   pp.SetSpeed(somatorio-> velocidade, somatorio -> angulo)  ;
```

---------- // ----------

---------- // ----------

---------- // ----------

A seguir é apresentado o código fonte completo da simulação completa.

```
//  Código Fonte Programação Baseada em Comportamento Utilizando
Arquitetura de Esquemas Motores

//  Fabiano Santério   2009


#include <includes/playerc++.h>
#include <iostream>
#include <time.h>             // Biblioteca da variável Tempo
#include <pthread.h>          // Biblioteca de Multi-tasking C++
#include "includes/args.h"

using namespace PlayerCc;

PlayerClient robot (gHostname, gPort);    // Inicializando o robo
Position2dProxy pp (&robot, gIndex);      // Inicializando função
de posição 2D
SonarProxy sp (&robot, gIndex);           // Inicializando
sensores de ultra som
BlobfinderProxy bp (&robot, gIndex);      // Inicializando sensor
de cores
GripperProxy gp (&robot, gIndex);         // Inicializando garra
BumperProxy bump (&robot, gIndex);        // Inicializando Bumper
LaserProxy lp(&robot, gIndex);            // Inicializando
Sensores a Laser


////////// Inicialização das Flags Dos Comportamentos //////////

int aquisitar_flag = 0;
int pegar_flag = 0;
int entregar_flag = 0;
int soltar_flag = 0;
```

```
////////////// Definindo Estruturas Básicas /////////

struct resultante {

      float velocidade;       // Magnitude Vetor Velocidade
      float angulo;                 // Ângulo
      float peso;             // Pesos dos Comportamentos
};

//////////////////  DEFININDO FUNÇÕES ////////////////////////

//  Função "lr_rot" retorna -1 ou 1  para virar à esquerda ou à
direita

int lr_rot(int lr_detect)     // lr_detect é o argumento da funcao
{
if (lr_detect == 1)           // se for esquerda
      return -1;        //  vira p/ direita
else
      return 1;         // se nao, vira p/ esq.
}

//  Função "bump_detect" detecta colisao na esqueda ,direita e
traseira
//  Retorna 1 esq,  2 dir,  3  traseira , 4 frontal  e  0 sem
colisao

double bump_detect_min_dist = 1.5;    //  distancia minima para
detecção de colisao

int bump_detect()
{

if      (sp[1] < bump_detect_min_dist || sp[2] <
bump_detect_min_dist)         // esquerda frontal
      return 1;
else if (sp[5] < bump_detect_min_dist || sp[6] <
bump_detect_min_dist)         // direita frontal
      return 2;
else if (sp[10] < bump_detect_min_dist || sp[11] <
bump_detect_min_dist || sp[12] < bump_detect_min_dist || sp[13] <
bump_detect_min_dist)   // traseira
      return 3;
else if (sp[3] < bump_detect_min_dist || sp[4] <
bump_detect_min_dist)         // frontal
      return 4;
else if (sp[0] < bump_detect_min_dist || sp[7] <
bump_detect_min_dist || sp[8] < bump_detect_min_dist || sp[15] <
bump_detect_min_dist)   // laterais
      return 5;
else
return 0;
}

///Definindo COMPORTAMENTOS PRIMITIVOS (Esquemas Perceptivos) ////

// ANDA E GIRA -> Anda em linha reta durante um determinado tempo
e gira em seu próprio eixo

double andagira_tempo_cru =  7;           // Tempo de Cruzeiro
```

```
double andagira_vel_cru =  0.6;           // Velocidade de
Cruzeiro
double andagira_tempo_rot =  5;           // Tempo de Rotação
double andagira_vel_rot =  1;       // Velocidade de Rotação
time_t andagira_tempo_inicio = 0;   // Contador de Tempo
Regressivo
int andagira_acao_atual = 1;        // Ultima Ação Executada   0 =
Andar  1 = Rodar

resultante* andagira ()
{
      time_t tempo_atual = time(NULL);
      double tempo_comparacao = (andagira_acao_atual == 0)?
andagira_tempo_cru : andagira_tempo_rot;

      //std::cout << difftime (tempo_atual,andagira_tempo_inicio)
<< std::endl;

      if (difftime (tempo_atual,andagira_tempo_inicio) >
tempo_comparacao)       // Decisão para troca de ação
      {
            andagira_acao_atual = (andagira_acao_atual == 0)? 1:0;
            andagira_tempo_inicio = time(NULL);

            andagira_vel_rot = (andagira_acao_atual == 0)?
andagira_vel_rot * -1 : andagira_vel_rot ;  // Inverte lado
rotação
      }
      resultante* andagira_vet = new resultante ();   // Criando o
Vetor de saída
      andagira_vet-> peso = 1;                  // Peso AndaGira

      if (andagira_acao_atual == 0)
      {
            andagira_vet-> velocidade = andagira_vel_cru;
            andagira_vet-> angulo = 0;
      }
      else
      {
            andagira_vet-> velocidade = andagira_vel_cru/2;
            andagira_vet-> angulo = andagira_vel_rot;
      }

            for (int i=0; i < bp.GetCount(); i++)     //  Desvia
do local de depósito amarelo
            {
                  if ( bp.GetBlob(i).id == 4 && bp.GetBlob(i).top
< 20) // Se objeto for amarelo e perto
                  {
                        andagira_vet-> velocidade =
andagira_vel_cru / 4;
                        andagira_vet-> angulo = 2*
andagira_vel_rot;
                  }
            }
      return andagira_vet;
}

// BUSCA -> Busca Objeto AZUL id = 1
// Campo Potencial Atrativo
```

```
double busca_vel_rot = 3.3;          // Velocidade de Rotação
double busca_pos_mediana = 40;       // Posição na qual o objeto se
encontra em frente ao robo
double busca_vel_cru = 0.7;          // Velocidade de Cruzeiro

resultante* busca()
{
      resultante* busca_vet = new resultante();
      busca_vet-> peso = 1;
      busca_vet-> velocidade = 0;
      busca_vet-> angulo = 0;
      aquisitar_flag = 0;

      for (int i=0; i < bp.GetCount(); i++)     //  Procura em
todos os objetos que encontrar
      {
            if ( bp.GetBlob(i).id == 1 )  // Se objeto for azul
            {
                  aquisitar_flag = 1;     // Ativa comportamento
AQUISITAR

                  if ( bp.GetBlob(i).top < 3 )  // Se estiver
muito próximo do objeto diminui a vel. até parar
                  {
                        busca_vet-> velocidade = (busca_vel_cru *
bp.GetBlob(i).top) / 3 ;
                        busca_vet-> angulo = busca_vel_rot *
((busca_pos_mediana- bp.GetBlob(i).x)/busca_pos_mediana);   //
(40-x)/40
                  }
                  else
                  {
                        busca_vet-> velocidade = busca_vel_cru ;
                        busca_vet-> angulo = busca_vel_rot *
((busca_pos_mediana- bp.GetBlob(i).x)/busca_pos_mediana);   //
(40-x)/40
                  }
            }
      }
      return busca_vet;
}

// ENTREGA -> Procura Local de Entrega (Objeto amarelo) id = 4
// Campo Potencial Atrativo

double entrega_vel_rot = 3;          // Velocidade de Rotação
double entrega_pos_mediana = 39;     // Posição na qual o objeto se
encontra em frente ao robo
double entrega_vel_cru = 0.8;        // Velocidade de Cruzeiro

resultante* entrega()
{
int garra_aux = gp.GetBeams();

      resultante* entrega_vet = new resultante();
      entrega_vet-> peso = 1;
      entrega_vet-> velocidade = entrega_vel_cru;

      for (int i=0; i < bp.GetCount(); i++)     //  Procura em
todos os objetos que encontrar
      {
```

```
            if ( bp.GetBlob(i).id == 4 && gp.GetState() == 2)
      // Se objeto for amarelo e Garra Fechada
            {
                  entregar_flag = 1;      // Ativa comportamento
ENTREGAR

                  if ( bp.GetBlob(i).top |= 0)  // Vai em direção
ao deposito amarelo estar sobre o mesmo
                  {
                        soltar_flag = 0;
                        entrega_vet-> velocidade =entrega_vel_cru;
                        entrega_vet-> angulo = entrega_vel_rot *
((entrega_pos_mediana- bp.GetBlob(i).x)/entrega_pos_mediana);   //
(40-x)/40
                              if ( (garra_aux |= 1) && (garra_aux
|= 2) && (garra_aux |= 3))    // Se objeto nao estiver dentro da
garra libera entregar e pegar
                                    {
                                    entregar_flag = 0;
                                    pegar_flag = 0;
                                    aquisitar_flag = 1;
                                    }
                  }
                  else        // Aciona Soltar e libera Entregar
                  {
                        soltar_flag = 1;
                        entregar_flag = 0;
                  }
            }
}
      return entrega_vet;
}
// FOGE - > Foge de objeto vermelho   id = 0
// Campo Potencial Repulsivo

double foge_vel_rot = 1;            // Velocidade de Rotação
double foge_min_dist = 3.9;         // Dist Min para detecção
double foge_vel_cru = 0.8;          // Velocidade de Cruzeiro

resultante* foge()
{
      resultante* foge_vet = new resultante();
      foge_vet-> peso = 1;                // Peso Foge
      foge_vet-> velocidade = 0;
      foge_vet-> angulo = 0;

      for (int i=0; i <= 360; i++)  //  Procura em toda a faixa de
leitura do sensor laser
      {
            if ( i <= 180 && lp[i] <= foge_min_dist)  // Se
inimigo no lado direito
            {
                  foge_vet-> velocidade = 2 * foge_vel_cru /
lp[i];
                  foge_vet-> angulo = (foge_vel_rot / ( (lp[i])  )
);
            }
            else if ( i >= 181 && i <= 360 && lp[i] <=
foge_min_dist) // Se inimigo no lado esquerdo
            {
```

```
                  foge_vet-> velocidade = 2 * foge_vel_cru /
lp[i];
                  foge_vet-> angulo = -(foge_vel_rot / ( (lp[i])
) ) ;
            }
      }
      return foge_vet;
}

//  DESVIA -> Desvia de Obstáculos Estáticos Utilizando Sonar
//  Campo Potencial Repulsivo

double desvia_limite_max_cru =  0.5;      // Limite Máximo de
Velocidade de Cruzeiro
double desvia_limite_max_rot =  0.6;      // Velocidade de Rotação
double desvia_dist_max = 4;               // Distância Máxima para
deteção de obstáculso
double desvia_dist_azul = 0.5;            // Distãncia na qual
desvia é desligado

float dist[16] ;                    // Vetor de variáveis de
contribuições para rotação

resultante* desvia()
{

      resultante* desvia_vet = new resultante();

      desvia_vet-> velocidade = 0;
      desvia_vet-> angulo = 0 ;
      desvia_vet-> peso = 1 ;

      for (int i = 0; i<16; i++)    // Roda em todos os sensores e
no vetor de contribuições de rotação
      {
            if (! (( (i == 3 || i == 4) < desvia_dist_azul ) &&
aquisitar_flag == 1))
            {
                  if (sp[i] < desvia_dist_max)
                  {
                        desvia_vet-> velocidade += 1/
((sp[i]*sp[i])*8);      // Desvia com quadrado da distãcia
                        desvia_vet-> angulo += dist[i]/
(sp[i]*sp[i]*0.1);      // Desvia com quadrado da distãcia
                  }
            }
            else
            {
            desvia_vet-> peso = 0 ;
            }
}
      if (desvia_vet-> velocidade > desvia_limite_max_cru)  //
Define limite máximo de velocidade e angulo
      {
            desvia_vet-> velocidade = desvia_limite_max_cru;
      }
      if (desvia_vet-> angulo > desvia_limite_max_rot ||
desvia_vet-> angulo < -desvia_limite_max_rot )
      {
            desvia_vet-> angulo = (( desvia_vet-> angulo > 0)? 1 :
-1) * desvia_limite_max_rot;
```

```
      }
      return desvia_vet;
}

//  COLISAO -> Evita Colisão
//  Campo Potencial Repulsivo

double colisao_vel_rot = 0.8;       // Velocidade de Rotação
double colisao_vel_cru = -0.3;            // Velocidade de Ré
double colisao_dist_azul = 2;

resultante* colisao()
{

      resultante* colisao_vet = new resultante();
      colisao_vet-> peso = 1;             // Peso Colisao

if (! (( (sp[3] || sp[4]) < colisao_dist_azul ) && aquisitar_flag
== 1))      // Desliga sensores 3 e 4 qd obj azul na frente
{
            int es_hit =  0;              // Variável Local para
detecção de objetos muito próximos */
            es_hit =  bump_detect();//  Chama Função Bumper Detect

      if (es_hit == 3)                    //  Colisão Traseira
      {
            colisao_vet-> velocidade = -2 * colisao_vel_cru;
      //  Anda um pouco pra frente
            colisao_vet-> angulo = 0;
      //  Nao gira
      }
      else if  (es_hit  ==  1  ||  es_hit == 2 || es_hit == 4)
      //  If Colisão Esquerda Direita ou Frontal
      {
            colisao_vet-> velocidade = colisao_vel_cru;
      //  Dá Ré
            colisao_vet-> angulo = lr_rot(es_hit) *
colisao_vel_rot;  //  Gira Sentido Contrário ao Obstáculo
      }
      else if  (es_hit  ==  5)
      //  If Colisão Lateral
      {
            colisao_vet-> velocidade = -5 * colisao_vel_cru;
      //  Acelera ao máximo
            colisao_vet-> angulo = 0;
      //  Não Gira
      }
}
else
{
colisao_vet-> peso = 0;
}
      return colisao_vet;
}

// RUIDO -> Gera Ruído Aleatório para evitar situações singulares

resultante* ruido ()
{
      resultante* ruido_vet = new resultante ();      // Criando o
Vetor de saída
```

```
      ruido_vet-> peso = 1;               // Peso Ruido

      float rd = rand() % 20 ;    // Numero aleatório entre 0 e 20

      ruido_vet-> velocidade = ((rd/(100)) - 0.05) ;  // Gerando
aleatório entre -0.05 e 0.15
      ruido_vet-> angulo = ((rd/(100)) - 0.1) ; // Gerando
aleatório entre -0.1 e 0.1

      return ruido_vet;
}

// FECHAR_garra -> Aciona a Garra para pegar objeto.

resultante* fechar_garra ()
{
      resultante* fechar_garra_vet = new resultante ();     //
Criando o Vetor de saída
      fechar_garra_vet-> peso = 1;                    // Peso
Fechar garra

// gp.GetBeams -> Sensor Frontal Atuado = 1 - Sensor Traseiro
Atuado = 2 - Ambos Atuados = 3
// gp.GetState -> Estado Aberto = 1  - Estado Fechado = 2 - Estado
Movendo = 3

    if ( gp.GetBeams() == 2 || gp.GetBeams() == 3)    // Se objeto
estiver dentro da garra
      {
      pegar_flag = 1;
        gp.Close();

            if ( gp.GetState() == 2 )     // Se Garra Fechada
Libera Flag Pegar e Aciona Flag Entregar
            {
                  pegar_flag = 0;
                  entregar_flag = 1;
            }
      }
      else
      {
        gp.Open();
      entregar_flag = 0;
      pegar_flag = 0;
      }

      fechar_garra_vet-> velocidade = 0.08;
      fechar_garra_vet-> angulo = 0;
      return fechar_garra_vet;
}

// ABRIR_garra -> Aciona a garra para soltar objeto

time_t abrir_garra_tempo_inicio = 0;      // Contador Tempo
double abrir_garra_tempo_cru = 0.8; // Tempo de entrada na zona de
depósito amarela
double abrir_garra_tempo_parada = 0.2;    // Tempo de parada na
zona de depósito amarela
double abrir_garra_tempo_rot = 1;   // Tempo de rotacao na zona de
depósito amarela
```

```
double abrir_garra_vel_rot = 1.8;   // Velocidade de rotacao na
zona de depósito amarela
double abrir_garra_vel_cru = 0.5;   // Velocidade de cruzeiro na
zona de depósito amarela
int abrir_garra_acao_atual = 2;     // Ultima Ação Executada   0 =
Andar  1 = Parar  2= Girar

resultante* abrir_garra ()
{
      resultante* abrir_garra_vet = new resultante ();      //
Criando o Vetor de saída
      abrir_garra_vet-> peso = 1;         // Peso Abrir garra

      if ( soltar_flag == 1 ) // Se flag acionado abre garra
      {
            entregar_flag = 0;
            aquisitar_flag = 0;
            pegar_flag = 0;

            time_t abrir_garra_tempo_atual = time(NULL);

            double abrir_garra_tempo_comparacao = 0;

            if ( abrir_garra_acao_atual == 0 )
            {
            abrir_garra_tempo_comparacao = abrir_garra_tempo_cru;
            }
            else if ( abrir_garra_acao_atual == 1 )
            {
            abrir_garra_tempo_comparacao =
abrir_garra_tempo_parada;
            }
            else
            {
            abrir_garra_tempo_comparacao = abrir_garra_tempo_rot;
            }

            if (difftime
(abrir_garra_tempo_atual,abrir_garra_tempo_inicio) >
abrir_garra_tempo_comparacao)       // Decisão para troca de ação
            {

                        if ( abrir_garra_acao_atual == 0 )
                        {
                        abrir_garra_acao_atual = 1;
                        }
                        else if ( abrir_garra_acao_atual == 1 )
                        {
                        abrir_garra_acao_atual = 2;
                        }
                        else
                        {
                        abrir_garra_acao_atual = 0;
                        }

                  abrir_garra_tempo_inicio = time(NULL);

            }
            if (abrir_garra_acao_atual == 0)
            {
```

```
                  abrir_garra_vet-> velocidade =
abrir_garra_vel_cru;
                  abrir_garra_vet-> angulo = 0;

                  if ( gp.GetState() == 1 )     // Se Garra Aberta
Libera Flag Soltar
                  {
                  soltar_flag = 0;
                  abrir_garra_acao_atual = 2;
                  }
                  else
                  {
                  soltar_flag = 1;
                  }
            }
            else if (abrir_garra_acao_atual == 1)
            {
                  abrir_garra_vet-> velocidade = 0;
                  abrir_garra_vet-> angulo = 0;
                  gp.Open();
            }
            else
            {
                  abrir_garra_vet-> velocidade = 0;
                  abrir_garra_vet-> angulo = abrir_garra_vel_rot;
            }
      }
return abrir_garra_vet;
}

// batida -> Executa manobra após detectar colisão
// 0 Dir     1 Esq     2 Tras


double batida_vel_rot = 5;          // Velocidade de Rotação
double batida_vel_cru = 4;          // Velocidade de Ré

resultante* batida ()
{
      resultante* batida_vet = new resultante ();     // Criando o
Vetor de saída
      batida_vet-> peso = 1;                    // Peso batida
      batida_vet-> velocidade = 0;
      batida_vet-> angulo = 0;

      if (bump.IsAnyBumped())
      {
            batida_vet-> velocidade = batida_vel_cru;

            if (bump.IsBumped(0))                     // Se bater
no tras. dir. anda pra frente e gira pra esq.
            {
                  batida_vet-> angulo = batida_vel_rot;
            }
            else if ((bump.IsBumped(1)))              // Se bater
no tras. esq. anda pra frente e gira pra dir.
            {
                  batida_vet-> angulo = -batida_vel_rot;
            }
            else if ((bump.IsBumped(2)))              // Se bater
no traseiro anda pra frente apenas
```

```
            {
                  batida_vet-> angulo = 0;
            }
            else if ((bump.IsBumped(3)))              // Se bater
no lateral dir. recua e gira pra esq.
            {
                  batida_vet-> angulo = batida_vel_rot;
                  batida_vet-> velocidade = -batida_vel_cru;
            }
            else                                    // Se bater no
lateral esq. recua e gira pra dir.
            {
                  batida_vet-> angulo = -batida_vel_rot;
                  batida_vet-> velocidade = -batida_vel_cru;
            }
      }
      return batida_vet;
}

////////////////////////////////////////////////////////////////
////////////////////// FUNÇÃO PRINCIPAL ////////////////////////

int main(void)
{
      robot.Read();     //Faz a primeira varredura nos sensores

//    Definindo Parâmetros
      dist[0] = -0.1; dist[1] = -0.2;     dist[2] = -0.3;
      dist[3] = -0.4;   dist[4] = 0.4;    dist[5] = 0.3;
      dist[6] = 0.2;    dist[7] = 0.1;    dist[8] = 0.1;
      dist[9] = 0.2;    dist[10] = 0.3;   dist[11] = 0.4;
      dist[12] = -0.4;dist[13] = -0.3;    dist[14] = -0.2;
      dist[15] = -0.1;

      for(;;)
      {

   robot.Read();  //Faz leitura de todos os sensores

   // Chamando os esquemas motores

   resultante* andagira_vet = andagira ();
// Anda durante um tempo e gira em torno do proprio eixo

   resultante* busca_vet = busca ();
// QUando detecta objetivo por cor , vai em sua direção

   resultante* entrega_vet = entrega ();
// Quando detecta local de entrega por cor, vai em sua direção

   resultante* foge_vet = foge ();
// Quando detecta inimigo por cor, desvia

   resultante* desvia_vet = desvia ();
// Desvia de obstáculo usando Sonar

   resultante* colisao_vet = colisao ();
// Evita colisões

   resultante* ruido_vet = ruido ();
// Ruído Aleatório
```

```
   resultante* fechar_garra_vet = fechar_garra ();
// Fechar Garra

   resultante* abrir_garra_vet = abrir_garra ();
// Abrir Garra

   resultante* batida_vet = batida ();
// Batida

/////////////// Definindo Comportamentos  ////////////////////////

   // Comportamento

   resultante* somatorio = new resultante();

   somatorio-> velocidade = 0;
   somatorio-> angulo =  0 ;

////////// Arbitrador   //////////

      if (soltar_flag == 1)                    //  FOGE, RUIDO,
ABRIR GARRA
      {
      std::cout << "Comportamento Ativo ->  !! SOLTAR !! \n"  <<
std::endl;

      andagira_vet->peso = 0;
      busca_vet->peso = 0;
      entrega_vet->peso = 0;
      desvia_vet->peso = 0;
      colisao_vet->peso = 0;
      fechar_garra_vet->peso = 0;
      batida_vet->peso = 0;
      }
      else if (entregar_flag == 1)             // ENTREGA , FOGE
,DESVIA,COLISAO,RUIDO, BATIDA
      {
      std::cout << "Comportamento Ativo ->  !! ENTREGAR !!\n"  <<
std::endl;

      andagira_vet->peso = 0;
      busca_vet->peso = 0;
      fechar_garra_vet->peso = 0;
      abrir_garra_vet->peso = 0;
      }
      else if (pegar_flag == 1)
      {
      std::cout << "Comportamento Ativo ->  !! PEGAR !!\n"  <<
std::endl;        // FOGE,RUIDO,FECHAR GARRA , BATIDA

      andagira_vet->peso = 0;
      busca_vet->peso = 0;
      entrega_vet->peso = 0;
      desvia_vet->peso = 0;
      colisao_vet->peso = 0;
      abrir_garra_vet->peso = 0;
      }
      else if (aquisitar_flag == 1)
      {
```

```
      std::cout << "Comportamento Ativo ->  !! AQUISITAR !!\n"  <<
std::endl;  // BUSCA , FOGE , DESVIA , COLISAO , RUIDO , BATIDA

      andagira_vet->peso = 0;
      entrega_vet->peso = 0;
      fechar_garra_vet->peso = 0;
      abrir_garra_vet->peso = 0;
      }
      else
      {
      std::cout << "Comportamento Ativo ->  !! EXPLORAR !!\n"  <<
std::endl;  // ANDAGIRA , FOGE , DESVIA , COLISAO , RUIDO , BATIDA

      busca_vet->peso = 0;
      entrega_vet->peso = 0;
      fechar_garra_vet->peso = 0;
      abrir_garra_vet->peso = 0;
      }

  somatorio-> velocidade = 0
                   + andagira_vet->velocidade * andagira_vet->peso
                   + busca_vet->velocidade * busca_vet->peso
                   + entrega_vet->velocidade * entrega_vet->peso
                   + foge_vet->velocidade * foge_vet->peso
                   + desvia_vet->velocidade * desvia_vet->peso
                   + colisao_vet->velocidade * colisao_vet->peso
                   + ruido_vet->velocidade * ruido_vet->peso
                   + fechar_garra_vet->velocidade *
fechar_garra_vet->peso
                   + abrir_garra_vet->velocidade *
abrir_garra_vet->peso
                   + batida_vet->velocidade * batida_vet->peso
                                                          ;

  somatorio-> angulo = 0
                   + andagira_vet->angulo * andagira_vet->peso
                   + busca_vet->angulo * busca_vet->peso
                   + entrega_vet->angulo * entrega_vet->peso
                   + foge_vet->angulo * foge_vet->peso
                   + desvia_vet->angulo * desvia_vet->peso
                   + colisao_vet->angulo * colisao_vet->peso
                   + ruido_vet->angulo * ruido_vet->peso
                   + fechar_garra_vet->angulo * fechar_garra_vet-
>peso
                   + abrir_garra_vet->angulo * abrir_garra_vet-
>peso
                   + batida_vet->angulo * batida_vet->peso
                                                         ;
//// Definindo Limites de Velocidade e Taxa de Rotação
somatorio-> velocidade = (somatorio-> velocidade > 1)? 1 :
somatorio-> velocidade;
somatorio-> velocidade = (somatorio-> velocidade < -0.7)? -0.7 :
somatorio-> velocidade;
somatorio-> angulo = (somatorio-> angulo > 5)? 5 : somatorio->
angulo;
somatorio-> angulo = (somatorio-> angulo < -5)? -5 : somatorio->
angulo;

// Atuando no motores do robô

   pp.SetSpeed(somatorio-> velocidade, somatorio -> angulo)  ;
```

```
  std::cout << "Velocidade =" << somatorio-> velocidade << " -
Taxa Rotação = " << somatorio-> angulo << "\n\n" << std::endl;

  }
}
```

---------- // ----------

---------- // ----------

---------- // ----------

Apêndice III

Datasheets dos componentes utilizados nas simulações e experimentos:

- Motor *Integy Matrix Pro Lathe Motor 75T Single SCM7501*

***Matrix Pro Lathe Motor 75T Single SCM7501**

Product Description

Product Description

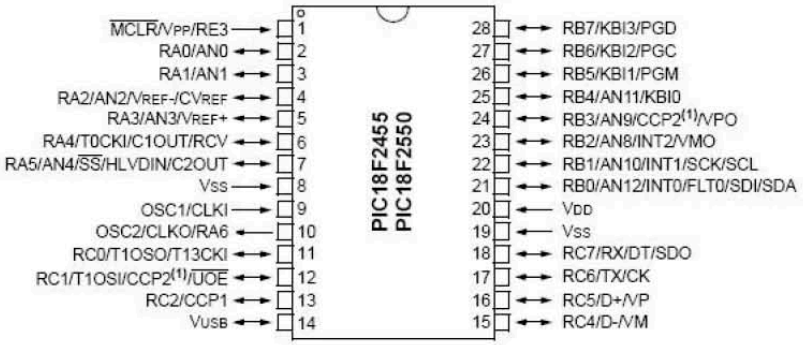
This is the 75T Pro Lathe Motor by Team Matrix. / / FEATURES: Low speed motor for lathe application / Ball bearings / Rebuildable motor design, endbell can be removed with removal of / two screws / / INCLUDES: One 75T Pro Lathe motor by Team Matrix / / REQUIRES: INTC1394 (motor lathe) / / SPECS: Length: 2-5/8"" (64mm) / Diameter: 1-1/2"" (36mm) / Shaft Diameter: 1/8"" (3mm) / / ajw 10/8/06 / ir/jxs

- Micro controlador PIC2550.

PIC18F2455/2550

Pin Diagrams

28-Pin PDIP, SOIC



PIC18F2550 Status: In Production

PIC18F2455/2550/4455/4550 Data sheet (10/27/2009)

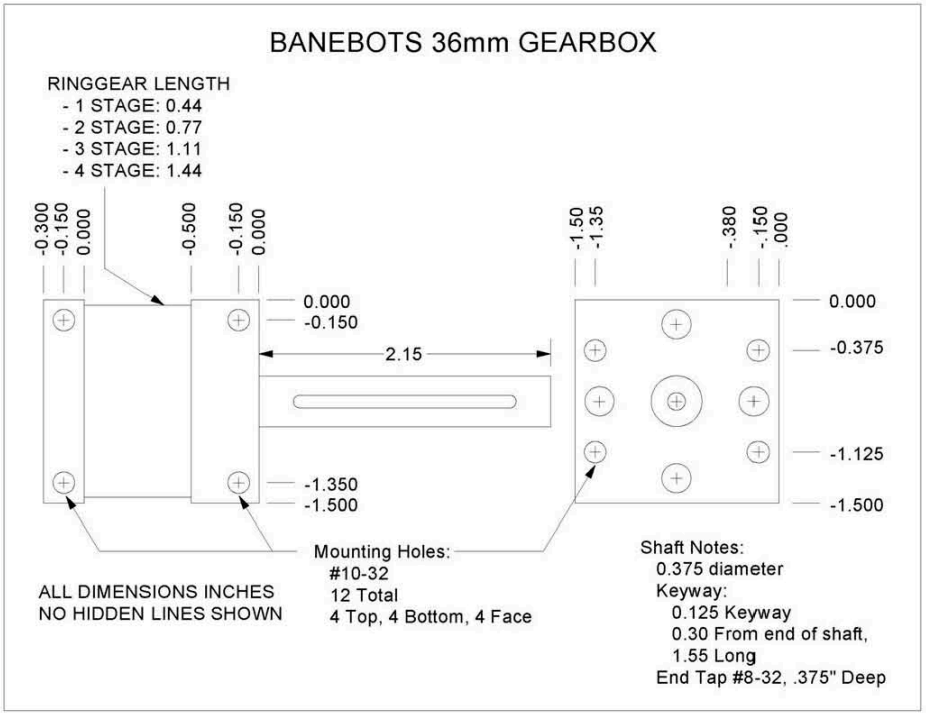
Ideal for low power (nanoWatt) and connectivity applications that benefit from the availability of three serial ports: FS-USB (12 Mbit/s), I²C™ and SPI™ (up to 10Mbit/s) and an asynchronous (LIN capable) serial port (EUSART). Large amounts of RAM memory for buffering and Enhanced FLASH program memory make it ideal for embedded control and monitoring applications that require periodic connection with a (legacy free) Personal Computer via USB for data upload/download and/or firmware updates. While operating up to 48 MHz, the PIC18F2550 is also mostly software and hardware compatible with the PIC16C745 Low-Speed USB OTP devices. THE PICSTART® Plus does NOT currently support this device but may support it in the future.
USB Application Design Center

Parameter Name	Value
Program Memory Type	Flash
Program Memory (KB)	32
CPU Speed (MIPS)	12
RAM Bytes	2,048
Data EEPROM (bytes)	256
Digital Communication Peripherals	1-A/E/USART, 1-MSSP(SPI/I2C)
Capture/Compare/PWM Peripherals	2 CCP
Timers	1 x 8-bit, 3 x 16-bit
ADC	10 ch, 10-bit
Comparators	2
USB (ch, speed, compliance)	1, Full Speed, USB 2.0
Temperature Range (C)	-40 to 85
Operating Voltage Range (V)	2 to 5.5
Pin Count	28

Features

Full Speed USB 2.0 (12Mbit/s) interface
- 1K byte Dual Port RAM + 1K byte GP RAM
- Full Speed Transceiver
- 16 Endpoints (IN/OUT)
- Internal Pull Up resistors (D+/D-)
- 48 MHz performance (12 MIPS)
- Pin-to-pin compatible with PIC16C7X5

- Caixa de redução *BaneBots* 36mm 16:1

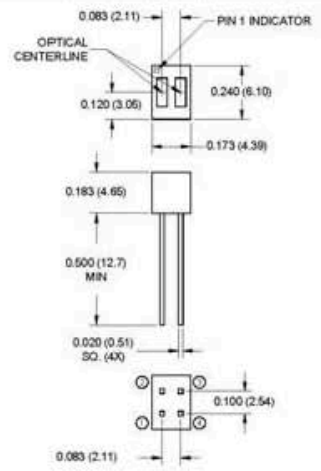


- Sensor infravermelho *FairChild* QRD1114.

FAIRCHILD
SEMICONDUCTOR

QRD1113/1114
REFLECTIVE OBJECT SENSOR

PACKAGE DIMENSIONS



PIN 1 COLLECTOR PIN 3 ANODE
PIN 2 EMITTER PIN 4 CATHODE

NOTES:
1. Dimensions for all drawings are in inches (millimeters).
2. Tolerance of ± .010 (.25) on all non-nominal dimensions unless otherwise specified.
3. Pins 2 and 4 typically .050" shorter than pins 1 and 3.
4. Dimensions controlled at housing surface.

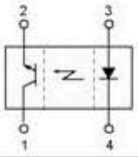
FEATURES

- Phototransistor Output
- No contact surface sensing
- Unfocused for sensing diffused surfaces
- Compact Package
- Daylight filter on sensor

NOTES (Applies to Max Ratings and Characteristics Tables.)

1. Derate power dissipation linearly 1.33 mW/°C above 25°C.
2. RMA flux is recommended.
3. Methanol or isopropyl alcohols are recommended as cleaning agents.
4. Soldering iron 1/16" (1.6mm) from housing.
5. As long as leads are not under any spring tension.
6. D is the distance from the sensor face to the reflective surface.
7. Cross talk (I_{CX}) is the collector current measured with the indicator current on the input diode and with no reflective surface.
8. Measured using an Eastman Kodak neutral white test card with 90% diffused reflecting as a reflective surface.

SCHEMATIC



ABSOLUTE MAXIMUM RATINGS (T_A = 25°C unless otherwise specified)

Parameter	Symbol	Rating	Units
Operating Temperature	T_{OPR}	-40 to +85	°C
Storage Temperature	T_{STG}	-40 to +85	°C
Lead Temperature (Solder Iron)[2,3]	T_{SOL-I}	240 for 5 sec	°C
Lead Temperature (Solder Flow)[2,3]	T_{SOL-F}	260 for 10 sec	°C
EMITTER			
Continuous Forward Current	I_F	50	mA
Reverse Voltage	V_R	5	V
Power Dissipation[1]	P_D	100	mW
SENSOR			
Collector-Emitter Voltage	V_{CEO}	30	V
Emitter-Collector Voltage	V_{ECO}		V
Power Dissipation[1]	P_D	100	mW

FAIRCHILD
SEMICONDUCTOR

QRD1113/1114
REFLECTIVE OBJECT SENSOR

ELECTRICAL / OPTICAL CHARACTERISTICS (T_A = 25°C)

PARAMETER	TEST CONDITIONS	SYMBOL	MIN	TYP	MAX	UNITS
EMITTER Forward Voltage	I_F = 20 mA	V_F	—	—	1.7	V
Reverse Current	V_R = 5 V	I_R	—	—	100	µA
Peak Emission Wavelength	I_F = 20 mA	λ_{PE}	—	940	—	nm
SENSOR Collector-Emitter Breakdown	I_C = 1 mA	BV_{CEO}	30	—	—	V
Emitter-Collector Breakdown	I_E = 0.1 mA	BV_{ECO}	5	—	—	V
Dark Current	V_{CE} = 10 V, I_F = 0 mA	I_D	—	—	100	nA
COUPLED QRD1113 Collector Current	I_F = 20 mA, V_{CE} = 5 V D = .050" (5,6)	$I_{C(ON)}$	0.300	—	—	mA
QRD1114 Collector Current	I_F = 20 mA, V_{CE} = 5 V D = .050" (5,6)	$I_{C(ON)}$	1	—	—	mA
Collector Emitter Saturation Voltage	I_F = 40 mA, I_C = 100 µA D = .050" (5,6)	$V_{CE(SAT)}$	—	—	0.4	V
Cross Talk	I_F = 20 mA, V_{CE} = 5 V, E_E = 0 (7)	I_{CX}	—	.200	10	µA
Rise Time	V_{CE} = 5 V, R_L = 100 Ω	t_r	—	10	—	µs
Fall Time	$I_{C(ON)}$ = 5 mA	t_f	—	50	—	µs

FAIRCHILD
SEMICONDUCTOR

QRD1113/1114
REFLECTIVE OBJECT SENSOR

TYPICAL PERFORMANCE CURVES

Fig. 1 Forward Voltage vs. Forward Current

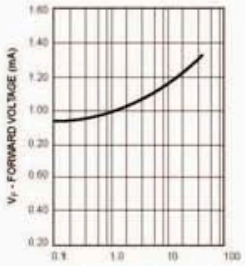


Fig. 2 Normalized Collector Current vs. Forward Current

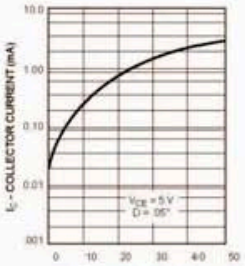


Fig. 3 Normalized Collector Current vs. Temperature

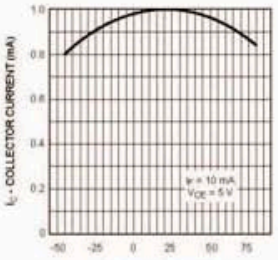


Fig. 4 Normalized Collector Dark Current vs. Temperature

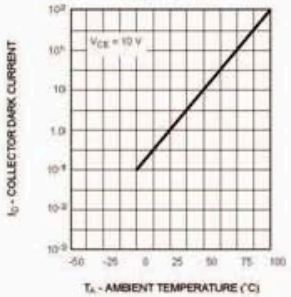
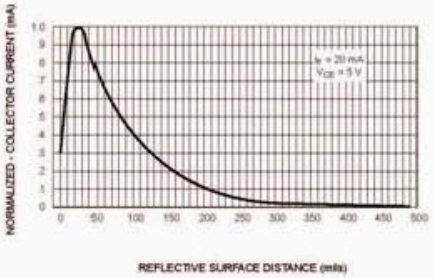


Fig. 5 Normalized Collector Current vs. Distance



FAIRCHILD
SEMICONDUCTOR

QRD1113/1114
REFLECTIVE OBJECT SENSOR

DISCLAIMER

FAIRCHILD SEMICONDUCTOR RESERVES THE RIGHT TO MAKE CHANGES WITHOUT FURTHER NOTICE TO ANY PRODUCTS HEREIN TO IMPROVE RELIABILITY, FUNCTION OR DESIGN. FAIRCHILD DOES NOT ASSUME ANY LIABILITY ARISING OUT OF THE APPLICATION OR USE OF ANY PRODUCT OR CIRCUIT DESCRIBED HEREIN; NEITHER DOES IT CONVEY ANY LICENSE UNDER ITS PATENT RIGHTS, NOR THE RIGHTS OF OTHERS.

LIFE SUPPORT POLICY

FAIRCHILD'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT OF FAIRCHILD SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and (c) whose failure to perform when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury of the user.
2. A critical component in any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

- Sensor Ultra som SRF10.

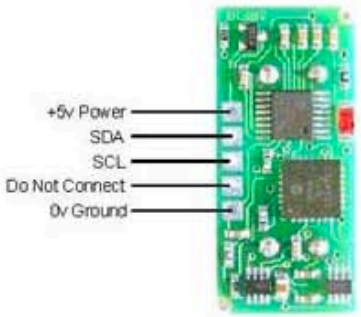
SRF10 Ultrasonic range finder

Technical Specification

Communication with the SRF10 ultrasonic rangefinder is via the I2C bus. This is available on popular controllers such as the OOPic and Stamp BS2p, as well as a wide variety of micro-controllers. To the programmer the SRF10 behaves in the same way as the ubiquitous 24xx series eeprom's, except that the I2C address is different. The default shipped address of the SRF10 is 0xE0. It can be changed by the user to any of 16 addresses E0, E2, E4, E6, E8, EA, EC, EE, F0, F2, F4, F6, F8, FA, FC or FE, therefore up to 16 sonar's can be used. We have examples of using the SRF10 module with a wide range of popular controllers.

Connections

The connections to the SRF10 are identical to the SRF08. The "Do Not Connect" pin should be left unconnected. It is actually the CPU MCLR line and is used once only in our workshop to program the PIC16F87 on-board after assembly, and has an internal pull-up resistor. The SCL and SDA lines should each have a pull-up resistor to +5v somewhere on the I2C bus. You only need one pair of resistors, not a pair for every module. They are normally located with the bus master rather than the slaves. The SRF10 is always a slave - never a bus master. If you need them, I recommend 1.8k resistors. Some modules such as the OOPic already have pull-up resistors and you do not need to add any more.



Registers

The SRF10 appears as a set of 4 registers.

Location	Read	Write
0	Software Revision	Command Register
1	Unused (reads 0x80)	Max Gain Register (default 16)
2	Range High Byte	Range Register (default 255)
3	Range Low Byte	N/A

Only locations 0, 1 and 2 can be written to. Location 0 is the command register and is used to start a ranging session. It cannot be read. Reading from location 0 returns the SRF10 software revision. By default, the ranging lasts for 65mS, but can be changed by writing to the range register at location 2. The SRF10 will not respond to commands on the I2C bus whilst it is ranging. See the **Changing Range** and

Analogue Gain sections below.

Locations, 2 and 3, are the 16bit unsigned result from the latest ranging - high byte first. The meaning of this value depends on the command used, and is either the range in inches, or the range in cm or the flight time in uS. A value of 0 indicates that no objects were detected.

Commands

The are three commands to initiate a ranging (80 to 82), to return the result in inches, centimeters or microseconds. There is also a set of commands to change the I2C address.

Command		Action
Decimal	Hex	
80	0x50	Ranging Mode - Result in inches
81	0x51	Ranging Mode - Result in centimeters
82	0x52	Ranging Mode - Result in micro-seconds
160	0xA0	1st in sequence to change I2C address
165	0xA5	3rd in sequence to change I2C address
170	0xAA	2nd in sequence to change I2C address

Ranging Mode

To initiate a ranging, write one of the above commands to the command register and wait the required amount of time for completion and read the result. The echo buffer is cleared at the start of each ranging. The default and recommended time for completion of ranging is 65mS, however you can shorten this by writing to the range register before issuing a ranging command.

Checking for Completion of Ranging

You do not have to use a timer on your own controller to wait for ranging to finish. You can take advantage of the fact that the SRF10 will not respond to any I2C activity whilst ranging. Therefore, if you try to read from the SRF10 (we use the software revision number a location 0) then you will get 255 (0xFF) whilst ranging. This is because the I2C data line (SDA) is pulled high if nothing is driving it. As soon as the ranging is complete the SRF10 will again respond to the I2C bus, so just keep reading the register until its not 255 (0xFF) anymore. You can then read the sonar data. Your controller can take advantage of this to perform other tasks while the SRF10 is ranging.

Changing the Range

The maximum range of the SRF10 is set by an internal timer. By default, this is 65mS or the equivalent of 11 metres of range. This is much further than the 6 metres the SRF10 is actually capable of. It is possible to reduce the time the SRF10 listens for an echo, and hence the range, by writing to the range register at location 2. The range can be set in steps of about 43mm (0.043m or 1.68 inches) up to 11 metres.
The range is ((Range Register x 43mm) + 43mm) so setting the Range Register to 0 (0x00) gives a maximum range of 43mm. Setting the Range Register to 1 (0x01) gives a maximum range of 86mm. More usefully, 24 (0x18) gives a range of 1 metre and 93 (0x5D) is 4 metres. Setting 255 (0xFF) gives the original 11 metres (255 x 43 + 43 is 11008mm). There are two reasons you may wish to reduce the range.
1. To get at the range information quicker
2. To be able to fire the SRF10 at a faster rate.
If you only wish to get at the range information a bit sooner and will continue to fire the SRF10 at 65ms of slower, then all will be well. However if you wish to fire the SRF10 at a faster rate than 65mS, you will definitely need to reduce the gain - see next section.

The range is set to maximum every time the SRF10 is powered-up. If you need a different range, change it once as part of your system initialization code.

Analogue Gain

The analogue gain register sets the *Maximum* gain of the analogue stages. To set the maximum gain, just write one of these values to the gain register at location 1. During a ranging, the analogue gain starts off at its minimum value of 40. This is increased at approx. 96uS intervals up to the maximum gain setting, set by register 1. Maximum possible gain is reached after about 100mm (4inches) of range. The purpose of providing a limit to the maximum gain is to allow you to fire the sonar more rapidly than 65mS. Since the ranging can be very short, a new ranging can be initiated as soon as the previous range data has been read. A potential hazard with this is that the second ranging may pick up a distant echo returning from the previous "ping", give a false result of a close by object when there is none. To reduce this possibility, the maximum gain can be reduced to limit the modules sensitivity to the weaker distant echo, whilst still able to detect close by objects. The maximum gain setting is stored only in the CPU's RAM and is initialized to maximum on power-up, so if you only want do a ranging every 65mS, or longer, you can ignore the Range and Gain Registers. The Gain Register is set to 16 (a gain of 700) at power-up. This can be decreased as required.

Gain Register		Maximum Analogue Gain
Decimal	Hex	
0	0x00	Set Maximum Analogue Gain to 40
1	0x00	As above - Analogue Gain to 40
2	0x01	Set Maximum Analogue Gain to 50
3	0x02	Set Maximum Analogue Gain to 60
4	0x03	Set Maximum Analogue Gain to 70
5	0x04	Set Maximum Analogue Gain to 80
6	0x05	Set Maximum Analogue Gain to 100
7	0x06	Set Maximum Analogue Gain to 120
8	0x07	Set Maximum Analogue Gain to 140
9	0x08	Set Maximum Analogue Gain to 200
10	0x09	Set Maximum Analogue Gain to 250
11	0x0A	Set Maximum Analogue Gain to 300
12	0x0B	Set Maximum Analogue Gain to 350
13	0x0C	Set Maximum Analogue Gain to 400
14	0x0D	Set Maximum Analogue Gain to 500
15	0x0E	Set Maximum Analogue Gain to 600
16	0x0F	Set Maximum Analogue Gain to 700

Note that the relationship between the Gain Register setting and the actual gain is not a linear one. Also there is no magic formula to say "use this gain setting with that range setting". It depends on the size, shape and material of the object and what else is around in the room. Try playing with different settings until you get the result you want. If you appear to get false readings, it may be echo's from previous "pings", try going back to firing the SRF10 every 65mS or longer (slower).
If you are in any doubt about the Range and Gain Registers, remember they are automatically set by the SRF10 to their default values when it is powered-up. You can ignore and forget about them and the SRF10 will work fine, detecting objects up to 6 metres away every 65mS or slower.

LED

The red LED is used to flash out a code for the I2C address on power-up (see below). It also gives a brief flash during the "ping" whilst ranging.

Changing the I2C Bus Address

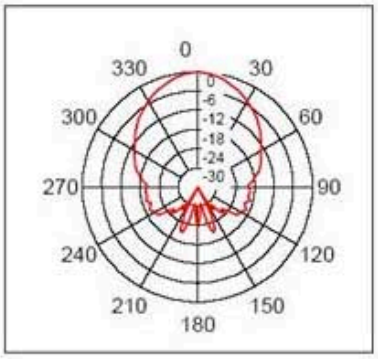
To change the I2C address of the SRF10 you must have only one sonar on the bus. Write the 3 sequence commands in the correct order followed by the address. Example; to change the address of a sonar currently at 0xE0 (the default shipped address) to 0xF2, write the following to address 0xE0; (0xA0, 0xAA, 0xA5, 0xF2). These commands must be sent in the correct sequence to change the I2C address, additionally, No other command may be issued in the middle of the sequence. The secuence must be sent to the command register at location 0, which means 4 separate write transactions on the I2C bus. When done, you should label the sonar with its address, however if you do forget, just power it up without sending any commands. The SRF10 will flash its address out on the LED. One long flash followed by a number of shorter flashes indicating its address. The flashing is terminated immediately on sending a command the SRF10.

Address		Long Flash	Short flashes
Decimal	Hex		
224	E0	1	0
226	E2	1	1
228	E4	1	2
230	E6	1	3
232	E8	1	4
234	EA	1	5
236	EC	1	6
238	EE	1	7
240	F0	1	8
242	F2	1	9
244	F4	1	10
246	F6	1	11
248	F8	1	12
250	FA	1	13
252	FC	1	14
254	FE	1	15

Take care not to set more than one sonar to the same address, there will be a bus collision and very unpredictable results.

Changing beam pattern and beam width

You can't! This is a question which crops up regularly, however there is no easy way to reduce or change the beam width that I'm aware of. The beam pattern of the SRF10 is conical with the width of the beam being a function of the surface area of the transducers and is fixed. It is possible to make the sonar less sensitive to objects off to the side by reducing the maximum gain register from 16 to a lower level. This is a the expense of shorter range, however most small robots don't need 6m of range. A value of 8 (max. gain 140) will reduce the practicable range to about 2m, but it will be much less sensitive to objects off the center line. The beam pattern of the transducers used on the SRF10, taken from the manufacturers data sheet, is shown below.



There is more information in the sonar faq.

Mounting the SRF10

You may have notice that there are no mounting holes on the SRF10 module! That was deliberate to keep the module as small as possible. So how do you mount it? Here are three suggestions:

1. A straight or right angle 0.1 inch connector soldered to your PCB.

2. Using two 9.5mm rubber grommets. Two holes should be drilled into the panel you're mounting the SRF10 to. The hole centers should be 0.7inches (17.78mm) apart and the holes drilled 0.5 inches (12.7mm) in diameter. The two grommets should then be fitted to the panel and the SRF10 gently pushed into them.

3. Using our SRF10 Mounting Kit, shown below.