# Programação Genética Distribuída: um algoritmo evolutivo embarcado para controlar uma população de robôs móveis

Anderson Luiz Fernandes Perez[1,2], Guilherme Bittencourt[2], and Mauro Roisenberg[3]

[1] Colegiado de Engenharia da Computação
Universidade Federal do Vale do São Francisco (UNIVASF)
Campus Juazeiro - BA - Brasil
anderson.perez@univasf.edu.br
[2] Departamento de Automação e Sistemas
Universidade Federal de Santa Catarina (UFSC)
Florianópolis - SC - Brasil
gb@das.ufsc.br
[3] Departamento de Informática e Estatística
Universidade Federal de Santa Catarina (UFSC)
Florianópolis - SC - Brasil
mauro@inf.ufsc.br

**Abstract.** This paper presents the DGP (Distributed Genetic Programming) algorithm an extension of classical genetic programming algorithm. DGP is applied in Embodied Evolution an Evolutionary Robotic technique that the evaluation, selection and reproduction are carried out by cooperation and competition of the robots, without any need for human intervention. To evaluate DGP's algorithm an experiment with box-pushing problem was accomplished. In this experiment a group of three robots needs to push one box putted in the middle of an environment. The experiment was accomplished in the Eyebot Simulator.

## 1 Introdução

Programar um robô para executar uma determinada tarefa, muitas vezes, exige que o programador tenha um bom conhecimento sobre o domínio do problema, ou seja, o programador é responsável por descrever todos os passos necessários que o robô deverá tomar para executar a tarefa.

Muitas vezes o programador não possui meios de prever possíveis problemas que o robô poderá enfrentar, principalmente se o ambiente no qual o robô irá atuar for dinâmico ou não estruturado. Nesse caso, é importante que o robô tenha um alto grau de autonomia e seja dotado de mecanismos que permitam sua auto-adaptação para poder tomar decisões em situações para as quais ele não foi programado.

Para tornar o sistema de controle de um robô móvel mais dinâmico, ou seja, adaptável, é necessário utilizar alguma técnica de desenvolvimento que permita que o sistema se modifique ao longo de sua execução. A Computação Evolutiva (CE), através de seus Algoritmos Evolutivos, possibilita o desenvolvimento de sistemas de controle adaptativos para robôs móveis.

A Programação Genética (PG) [4] é uma técnica da Computação Evolutiva que visa a geração automática de programas de computador. O principal objetivo da PG é ensinar computadores a se programar, isto é, a partir de especificações de comportamentos primários, o computador deve ser capaz de gerar um programa que satisfaça algumas condições que visam a solução de alguma tarefa ou problema.

Neste artigo é descrito o algoritmo da PGD (Programação Genética Distribuída), uma extensão do algoritmo clássico da programação genética. A PGD é aplicada na Evolução Embarcada, uma técnica da robótica evolutiva onde a avaliação, seleção e a reprodução são executadas pelos robôs de forma cooperativa e competitiva sem nenhuma intervenção humama. Este artigo está organizado como segue: na Seção 2 estão listados alguns trabalhos relacionados; na Seção 3 é descrito o funcionamento da PGD; a Seção 4 descreve um experimento com a PGD; a Seção 5 apresenta as conclusões e as perspectivas para trabalhos futuros.

## 2 Trabalhos Relacionados

Alguns experimentos com robôs móveis usando o algoritmo PEGA (*Physically Embedded Genetic Algorithm*) estão descritos em [6]. O objetivo dos experimentos foi a ativação de comportamentos individuais embarcados em cada robô. Os experimentos foram realizados com dois robôs móveis equipados com sensores de infra-vermelho, sensores de toque, sonar e sensores de luz. A comunicação entre os robôs era feita através de infra-vermelho.

Em [12] foram realizados experimentos para evoluir o sistema de controle de um grupo de 5 (cinco) robôs. O controle de cada robô foi implementado com uma rede neural do tipo RAM. As leituras de cada sensor eram convertidas em um sinal de 2 bits. A saída da rede consistia de 8 (oito) comandos para cada motor: S (*stop*), FS (*front slow*), FM (*front medium*), FF (*front fast*), TRS (*turn right sharp*), TRL (*turn right long*), TLS (*turn left sharp*) e TLL (*turn left long*).

Foi utilizado algoritmo genético para determinar o comando de saída para os motores, bem como para determinar a configuração sensorial dos robôs. Em cada robô dois genes determinam a presença ou a ausência de um determinado sensor. Essa configuração não somente faz com que o sistema de controle se adapte mas também permite alterar a morfologia de cada robô.

Em barlow2006-gecco e barlow2008-gecco é relatado um experimento usando programação genética para evoluir o sistema de controle de um veículo aéreo não tripulado. O experimento foi realizado em um ambiente simulado, um quadrado de 100 milhas náuticas em cada lado.

O objetivo da simulação era encontrar um conjunto de 10 (controladores) que pudessem ser comparados com outras técnicas afim de avaliar qual o me-

lhor controlador a ser utilizado em veículos aéreos não tripulados. Os resultados demonstraram que o melhor controlador gerado evolutivamente foi o melhor em todos os testes realizados mesmo quando perturbações eram inseridas no sistema.

## 3 Programação Genética Distribuída

A PGD (Programação Genética Distribuída) é uma extensão do algoritmo tradicional da PG. A PGD é baseada no Microbial GA [2], uma variação do AG, seu funcionamento é semelhante à recombinação (infecção) genética que acontece nas bactérias onde segmentos do DNA (*Deoxyribonucleic Acid - Ácido Desoxirribonucleico*) são transferidos entre dois membros da população.

Na PGD são considerados dois conjuntos de populações. O primeiro, chamado de conjunto local ou $P_{local_{R_i}}$, refere-se à população local de cada robô $R_i$, isto é, o conjunto de indivíduos ou soluções candidatas que estão embarcadas no robô. Cada $x \in P_{local_{R_i}}$ representa uma solução candidata a um problema. O segundo conjunto, chamado de conjunto total ou $P_{total}$, é formado pela união de todas as populações locais de cada robô, isto é, $P_{total} = P_{local_{R_1}} \cup P_{local_{R_2}} \cup P_{local_{R_3}} \cup \ldots, P_{local_{R_n}}$. O processo evolutivo ocorre sempre considerando a população total, ou seja, partes (sub-árvores) de um indivíduo local de um determinado robô podem ser consideradas no processo evolutivo da população local de outro robô.

Ao contrário de outras abordagens em Evolução Embarcada (EE), na PGD o processo evolutivo é assíncrono, isto é, não é necessário que dois robôs se sincronizem para se reproduzirem. Partes de um indivíduo mais adaptado são enviados para todos os outros robôs. A seqüência de passos do algoritmo da PGD é a seguinte [8]:

1. Criar aleatoriamente uma população de programas;
2. Executar iterativamente os seguintes passos até que algum critério de parada seja satisfeito:
   (a) Avaliar cada programa da população através de uma função de avaliação, que expressa a sua aptidão (*fitness*);
   (b) Receber uma mensagem $M$ [4] de um indivíduo remoto[5] enviadas por outro robô;
   (c) Selecionar os $t$ melhores indivíduos da população local usando o método de seleção por torneio [6];
   (d) Selecionar aleatoriamente uma parte do melhor indivíduo local (mais adaptado) e enviar uma mensagem $M$, contendo a parte selecionada mais o valor do *fitness*, em *broadcast* (difusão) para os outros robôs;
   (e) Comparar se o *fitness* do pior indivíduo selecionado localmente é menor que o *fitness* do indivíduo remoto. Se sim, executa o operador de mutação remota substituindo uma parte, selecionada aleatoriamente, do indivíduo local pela parte recebida do indivíduo remoto;

[4] Cada mensagem recebida contém o valor do *fitness* e uma parte da árvore do indivíduo remoto.
[5] Um indivíduo remoto é um programa que faz parte da população local de outro robô.
[6] O tamanho do torneio é um parâmetro definido antes da execução do algoritmo.

(f) Executar os operadores de cruzamento e mutação;

3. Retornar com o melhor programa encontrado.

No passo (d) do algoritmo da PGD, selecionar uma parte do melhor indivíduo local, refere-se a selecionar aleatoriamente uma função ou terminal que faz parte da estrutura do programa (indivíduo). Caso a parte selecionada seja uma função, então toda a estrutura dependente desta função, de acordo com o valor de sua aridade, deve ser também enviada na mensagem.

As partes recebidas remotamente são adicionadas a estrutura do pior indivíduo, dos $t$ melhores selecionados, obedecendo os critérios da Equação 1:

$$P_{local}(I) = \left\{ \begin{array}{ll} OMR(I) & \text{if } Fitness(I_{remoto}) > Fitness(I_{local}) \\ I & \text{if } Fitness(I_{remoto}) \leq Fitness(I_{local}) \end{array} \right\} \quad (1)$$

onde, $P_{local}(I)$ representa o indivíduo $I$ selecionado da população local ($P_{local}$). O operador de mutação remota ($OMR$) é executado sobre o indivíduo $I$ se o valor do *fitness* do indivíduo remoto ($Fitness(I_{remoto})$) for maior que o valor do *fitness* de $I$ ($Fitness(I_{local})$). Caso contrário, o indivíduo local $I$ não sofre a mutação e permanece com sua estrutura inalterada.

O método de seleção empregado na PGD é a seleção por torneio com a manutenção dos pais após o cruzamento. Esta é uma técnica elitista conhecida na bibliografia da área como *steady-state genetic programming*.

As mensagens trocadas entre os robôs devem conter o valor da aptidão e uma parte da estrutura que representa o indivíduo da população local. Para isso, todas as funções e terminais recebem uma identificação numérica única.

Para a execução da PGD, em cada robô, é necessário a existência de um sistema que dê suporte a execução e ao gerenciamento de todo o processo evolutivo, que ocorre de forma embarcada. O sistema desenvolvido com esse propósito chama-se SEGS (Sistema de Execução, Gerenciamento e Supervisão).

O SEGS é executado em cada robô que faz parte de um grupo de robôs, onde existe uma população local [7] que interage com a população local dos outros robôs. A cada geração, partes do melhor indivíduo de cada robô são enviadas (difusão) para todos os outros robôs do grupo.

O SEGS é estruturado em módulos que são ilustrados na Figura 1. Cada módulo é responsável por uma etapa do processo evolutivo.

Abaixo segue a descrição completa do objetivo e o funcionamento de cada módulo que compõe o SEGS, bem como a interligação entre eles [9]:

**Controle Evolutivo (CTE)**: é o componente principal do SEGS, é nele que está implementado a PGD. O CTE é responsável por criar, aleatoriamente, a população local de indivíduos através das bibliotecas de funções e terminais. A cada nova geração os indivíduos gerados são testados, isto é, executados pelo CTE. As FADs (Funções Automaticamente Definidas ou ADFs (*Automatically Defined Functions*) do Inglês) também são gerenciadas por esse componente. O

[7] A população local é um conjunto de programas candidatos a resolverem um problema. Esses programas estão embarcados no robô. Um problema é uma tarefa qualquer que o robô deve executar. Por exemplo, empurrar uma caixa.

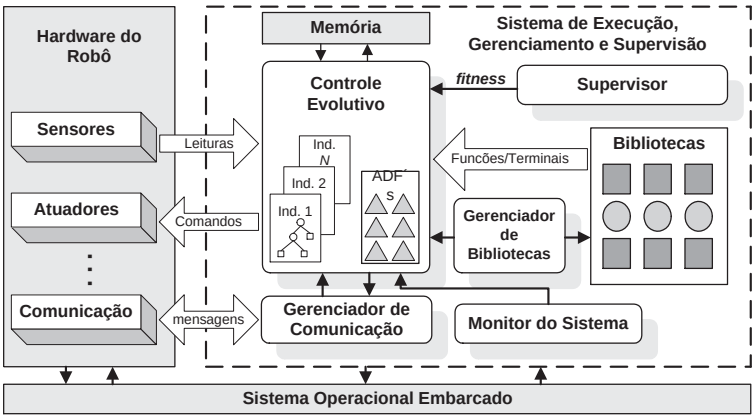


**Figura 1.** Estrutura modular do SEGS.

CTE está ligado ao Gerenciador de Comunicação, ao Gerenciador de Bibliotecas e à Memória.

**Memória**: o objetivo da memória é armazenar a representação dos indivíduos mais adaptados em cada geração para poder ser utilizada num processo de recuperação em caso de falhas do CTE ou até mesmo para otimizar tarefas que envolvem mais de uma competência, como por exemplo, desviar de obstáculos e deslocar um objeto de um lugar para outro. Nesse caso, a memória funciona como uma espécie de "fotografia" do sistema, podendo ser utilizada para acelerar o processo de aprendizagem através de experiências realizadas no passado.

**Gerenciador de Comunicação (GC)**: é o componente responsável pelo envio e o recebimento das mensagens que são trocadas entre os robôs. No processo de envio de mensagens, o CTE repassa para o GC a seqüência de funções e terminais e o valor do *fitness* que serão enviados para os outros robôs. O GC cria uma mensagem contendo essas informações e a envia para os outros robôs. Na recepção, o GC recebe as mensagens enviadas pelos outros robôs e as repassa para o CTE.

**Gerenciador de Bibliotecas (GB)**: esse componente gerencia os conjuntos de terminais e funções de cada robô. Nesse componente todas as funções e terminais são identificadas por um identificador numérico único para poderem ser enviadas para outros robôs pelo GC. Ter as bibliotecas separadas do sistema evolutivo representa uma vantagem adicional, pois é possível, a qualquer momento adicionar ou remover funções ou terminais sem a necessidade de redefinições no sistema de controle. Por exemplo, caso o robô receba um conjunto de sensores novos com novas funcionalidades, basta adicionar essas informações ao conjunto de terminais nas bibliotecas do SEGS. Isso faz com que os indivíduos gerados pelo CTE se adaptem às mudanças de características no hardware do robô (morfologia).

**Supervisor**: é responsável por avaliar e atribuir um valor de *fitness* para cada indivíduo, isso é feito através de um método de punição e recompensa.

Para um correto funcionamento desse método, para cada tarefa a ser realizada pelo robô, deve ser definido como e quando acontece a recompensa e a punição. Por exemplo, se a tarefa é a navegação livre de colisões, a recompensa e a punição podem ser definidas de acordo com a quantidade de colisões do robô. A cada choque com um obstáculo, o valor do *fitness* é decrementado (punição), caso contrário, o valor vai sendo incrementado de tempos em tempos (recompensa).

**Monitor**: é o componente responsável por avaliar a execução do sistema. Caso o sistema fique inativo, isto é, o robô fique parado por um longo período de tempo, o monitor ativa um processo de re-inicialização do CTE. Quando acontece a reinicialização do CTE, a imagem do melhor indivíduo, que está armazenada na memória, é recuperada e o processo evolutivo inicia-se a partir deste indivíduo, isto é, a população local é completada a partir deste indivíduo ao invés de começar de uma população aleatória como acontece no algoritmo tradicional da PG. Essa operação é realizada com o auxílio da função *Headless Chicken Crossover* (HCC) [10]. A HCC é uma operação de cruzamento onde somente um dos pais é escolhido da população, o outro é criado de forma aleatória toda a vez que a função é executada.

A PGD e o SEGS foram desenvolvidos em linguagem de programação C. A PGD foi implementada como uma biblioteca estática baseada no conceito de Programação Genética Linear [1]. A estrutura utilizada para a representação dos indivíduos (programas) é um vetor de tamanho $N$, sendo $N$ um parâmetro definido antes da execução do sistema.

## 4 Descrição do Experimento: empurrar uma caixa

Neste experimento um grupo de 3 (três) robôs precisam empurrar uma caixa, localizada no centro do ambiente, para qualquer direção. O experimento foi realizado no simulador do robô Eyebot [3]. A Figura 2 ilustra a interface do simulador Eyebot bem como a disposição dos robôs e da caixa no ambiente.

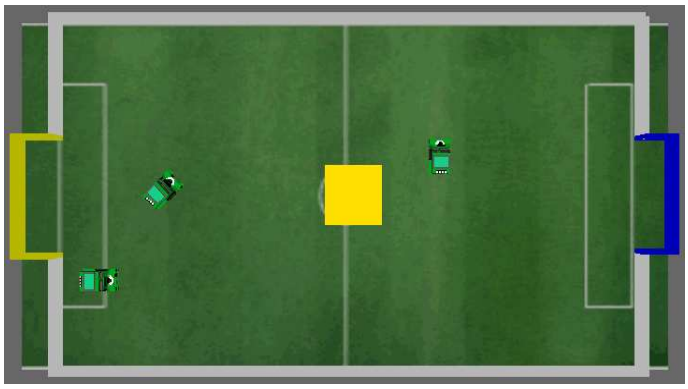

**Figura 2.** Interface principal do simulador Eyebot.

O conjunto de funções e de terminais utilizados nesse experimento estão descritos na Tabela 1. A coluna *Id.* representa a identificação de cada função e cada terminal.

**Tabela 1.** Conjunto de funções e de terminais para o exemplo 3.

| **Funções** | | | |
|---|---|---|---|
| **Nome** | **Aridade** | **Id.** | **Definição** |
| Prog1 | 1 | 2 | Executa um ramo da árvore. |
| Prog2 | 2 | 4 | Executa dois ramos da árvore. |
| Prog3 | 3 | 6 | Executa três ramos da árvore. |
| **Terminais** | | | |
| TurnRight | 0 | 1 | Virar a direita (15 graus). |
| TurnLeft | 0 | 3 | Virar a esquerda (15 graus). |
| GoForward | 0 | 5 | Seguir em frente (300 ms). |
| Return | 0 | 7 | Retornar (300ms). |
| BoxPushing | 0 | 9 | Empurrar a caixa. |
| TurnAroundRight | 0 | 11 | Girar para o lado direito. |
| TurnAroundLeft | 0 | 13 | Girar para o lado esquerdo. |

O terminal *BoxPushing* (Empurrar a Caixa) é o responsável por detectar a localização da caixa. Para isso, nele está implementado uma rotina de detecção de cor que é utilizada para detectar a posição da caixa. Quando a cor alaranjada estiver sendo detectada pela câmera, o terminal Empurrar a Caixa, faz com que o robô vá em direção a ela, no caso, em direção a caixa, fazendo com que o mesmo a desloque de posição.

Os seguintes parâmetros foram utilizados no problema de empurrar a caixa: *número de robôs: 3*; *Número de gerações: 50*; *tamanho da população: 5 indivíduos*; *tamanho do vetor: 60 posições*; *probabilidade de cruzamento: 60%*; *probabilidade de mutação: 10%*; *seleção por torneio com tamanho 2*; *número de execuções: 5*; *função de avaliação: método de punição e recompensa: se BOX-PUSHING; fitness += 5; se SE CHOCOU COM OBSTÁCULOS; fitness -= 1; senão fitness += 2*.

Os parâmetros definidos na simulação foram escolhidos baseados em execuções de teste. Nestas excuções foram testados diferentes parâmetros objetivando avaliar o comportamento do algoritmo da PGD. Os principais parâmetros testados foram a probabilidade de cruzamento e a probabilidade de mutação.

A função de avaliação é baseada no método de punição e recompensa, isto é, sempre que o robô encontrar a caixa e conseguir empurrá-la o valor da aptidão (*fitness*) é incrementado em 5. No caso de um choque com obstáculos, paredes ou outros robôs, a aptidão é decrementada em 1, caso contrário será incrementado em 2.

É importante ressaltar que o valor da aptidão não é incrementado por tempo, ou seja, a cada intervalo de tempo em que o robô navegar sem que se choque com um obstáculo o valor da aptidão será incrementado. O valor da aptidão somente será incrementado ou decrementado quando os sensores forem lidos, no caso do Eyebot, três sensores de infravermelho e uma câmera.

As Figuras 3 e 4 ilustram, respectivamente, os gráficos com o valor médio da aptidão por geração e o valor da aptidão do melhor indivíduo por geração.

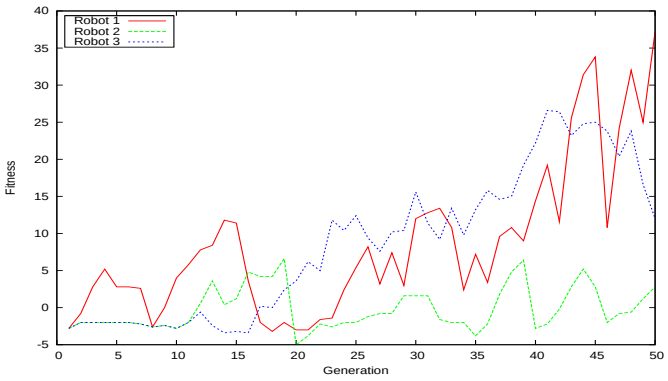


**Figura 3.** Média do valor de aptidão da população em cada geração para cada robô.

Durante a simulação todos os três robôs conseguiram tocar/empurrar a caixa. Entretanto, os robôs 1 e 3 tiveram um desempenho melhor, sendo que ambos conseguiram empurrar a caixa por um longo período de tempo, o que resultou num melhor valor de aptidão e conseqüentemente numa melhor média de aptidão população local de cada robô.

O robô 2 conseguiu empurrar a caixa algumas vezes, mas na média geral teve um desempenho inferior aos outros dois. Mesmo tendo influência dos melhores indivíduos dos outros dois robôs, ainda assim não foi o suficiente para acompanhar a média global do valor de aptidão. Este fato deu-se pois o robô 2 quase sempre ficava preso as laterais do ambiente simulado.

No gráfico da Figura 4 é possível perceber que a curva gerada pelo valor de aptidão do melhor indivíduo por geração de cada robô não difere muito da curva do gráfico da Figura 3, com os valores médios por população. O fato da PGD ser um algoritmo elitista, isso é, reproduzindo o melhor indivíduo de uma geração para outra, faz com que a melhor configuração para uma solução seja mantida e compartilhada entre os robôs da população de robôs.

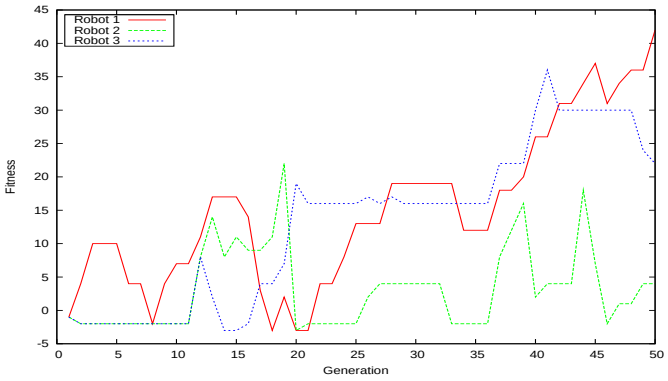


**Figura 4.** Valor de aptidão do melhor indivíduo em cada geração de cada robô.

## 5 Conclusão e Trabalhos Futuros

Este artigo apresentou o algoritmo da PGD (Programação Genética Distribuída) que é uma extensão do algoritmo clássico da programação genética. A PGD é aplicada na Evolução Embarcada, uma técnica da robótica evolutiva onde a avaliação, seleção e a reprodução são executadas pelos robôs de forma cooperativa e competitiva sem nenhuma intervenção humama. Com o propósito de avaliar a PGD foi realizado um experimento com o problema de empurrar uma caixa.

O experimento foi realizado com um grupo de três robôs em ambiente simulado. O principal objetivo era o deslocamento da caixa, situada inicialmente no centro do ambiente, para qualquer direção por todos os robôs do grupo. O experimento demonstrou que a PGD fez com que todos os robôs evoluissem seus sistemas de controle para atingir o objetivo de deslocar a caixa.

A principal contribuição deste trabalho está no desenvolvimento de um algoritmo evolutivo distribuído baseado em programação genética para ser aplicado na evolução do sistema de controle em populações de robôs móveis. A PGD difere de outras soluções em evolução embarcada (ver Seção 2 (Trabalhos Relacionados)) pois trata de evolução do sistema de controle de um conjunto de robôs móveis de maneira assíncrona e também pela forma com que representa os indivíduos da população local de cada robô.

Como trabalhos futuros pretende-se refazer o experimento de empurrar uma caixa, bem como outros dois experimentos realizados também em simulador relatados em [8] e [9], nos robôs Eyebot. O objetivo é confrontar os resultados obtidos em simulação com os resultados obtidos com o uso dos robôs reais.

## 6 Nota

O autor Guilherme Bittencourt faleceu durante o desenvolvimento deste trabalho.

## 7 Agradecimentos

O autor, Anderson Luiz Fernandes Perez, agradece o Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) pelo suporte financeiro.

## Referências

1. Broodier, M. and Banzhaf, W. (2006). *Linear Genetic Programming*. Springer.
2. Harvey, I. (2001). Artificial evolution: A continuing saga in evolutionary robotics: From intelligent robots to artificial life. In in Computer Science LNCS, S.-V. L. N., editor, *8th International Symposium on Evolutionary Robotics (ER2001)*.
3. Koestler, A. and Bräunl, T. (2004). Mobile robot simulation with realistic error models. In *2nd International Conference on Autonomous Robots and Agents*, pages 46–50, Palmerston North, New Zealand.
4. Koza, J. R. (1992). *Genetic Programming: A Paradigm for Genetically Breeding Computer Population of Computer Programs to Solve Problems*. MIT Press.
5. Mondada, F., Franzi, E., and Ienne, P. (1993). Mobile robot miniaturisation: A tool for investigation in control algorithms. In Springer-Verlag, editor, *of the 3rd International Symposium on Experimental Robotics*, Berlin.
6. Nehmzow, U. (2002). Physically embedded genetic algorithm learning in multi-robot scenarios: The pega algorithm. In *Second International Conference on Epigenetic Robotics*, pages 115–123, Edinburgh, Scotland.
7. Nordin, P. and Banzhaf, W. (1997). An on-line method to evolve behavior and to control a miniature robot in real time with genetic programming. *Adaptive Behaviour*, 5(2):107–140.
8. Perez, A. L. F., Bittencourt, G., and Roisenberg, M. (2008a). Embodied evolution with a new genetic programming variation algorithm. In *ICAS 2008 - Fourth International Conference on Automatic and Autonomous Systems*, pages 118–123, Gosier, Guadeloupe. IEEE Computer Society Press.
9. Perez, A. L. F., Bittencourt, G., and Roisenberg, M. (2008b). A new approach to control a population of mobile robots using genetic programming. In *SAC 2008 - Symposium on Applied Computing*, pages 1602–1606, Fortaleza, CE, Brazil.
10. Poli, R. and McPhee, N. F. (2001). Exact GP schema theory for headless chicken crossover and subtree mutation. In *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, pages 1062–1069, COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea. IEEE Press.
11. Sim, K.-B., Lee, D. W., and Zhang, B.-T. (2002). Behavior evolution of autonomous mobile robot (amr) using genetic programming based on evolvable hardware. *International Journal of Fuzzy Logic and Intelligent Systems*, 2:20–25.
12. Simões, E.V., K. R. D. (1999). An evolutionary controller for autonomous multirobot systems. In *International Conference on Systems, Man and Cybernetics*, pages 664–669, Tokyo, Japan. IEEE Press.
13. Barlow, Gregory J., Oh, Choong K. (2006). Robustness Analysis of Genetic Programming Controllers for Unmanned Aerial Vehicles. *Proceedings of the 2006 Genetic and Evolutionary Computation Conference*, pages 135–142, Seattle, WA.
14. Barlow, Gregory J. Oh, Choong K. Oh. (2008) Evolved Navigation Control for Unmanned Aerial Vehicles. *Frontiers in Evolutionary Robotics*, 20:353–378, Vienna, I-Tech Education and Publishing.