CENTRO UNIVERSITÁRIO DE BRASÍLIA – UniCEUB

CURSO DE ENGENHARIA DE COMPUTAÇÃO

Marcus Vinícius Sousa Leite de Carvalho

Robótica orientada espacialmente utilizando Algoritmos Genéticos

Orientador: Prof. Thiago de Miranda Leão Toribio

Brasília

Julho, 2010

Marcus Vinícius Sousa Leite de Carvalho

Robótica orientada espacialmente utilizando Algoritmos Genéticos

Trabalho apresentado ao Centro Universitário de Brasília (UniCEUB) como pré-requisito para a obtenção de Certificado de Conclusão de Curso de Engenharia de Computação.

Orientador: Prof. Thiago de Miranda Leão Toribio

Brasília

Julho, 2010

Marcus Vinícius Sousa Leite de Carvalho

Robótica orientada espacialmente utilizando Algoritmos Genéticos

Trabalho apresentado ao Centro Universitário de Brasília (UniCEUB) como pré-requisito para a obtenção de Certificado de Conclusão de Curso de Engenharia de Computação.

Orientador: Prof. Thiago de Miranda Leão Toribio

Este Trabalho foi julgado adequado para a obtenção do Título de Engenheiro de Computação, e aprovado em sua forma final pela Faculdade de Tecnologia e Ciências Sociais Aplicadas – FATECS.

_____

Prof. Abiezer Amarilia Fernandez

Coordenador do Curso

Banca Examinadora:

_____

Prof. Thiago de Miranda Leão Toribio, Mestre em Física

Orientador

_____

Prof. Flavio Antonio Klein, Mestre em Estatístaica e Métodos Quantitativos

Instituição

_____

Prof. Gil Renato Ribiero Gonçalves, Doutor em Física

Instituição

_____

Prof. Maria Marony Souza Farias Nascimento, Engenheira Elétrica

Instituição

AGRADECIMENTOS

Agradeço a minha família por todo incentivo e dedicação ao longo dessa jornada acadêmica.

Agradeço também aos meus colegas e amigos, Alana Ferrari, Daniel de Pilla Montibello, Débora Gadelha, Erismar de Moura, Robson Caetano, Felipe de Souza, Rômulo Eduardo, Gustavo Maciel, Gabriel Cordeido, Pedro Lacerda e Renan Humberto que me ajudaram bastante no incentivo a continuidade desse projeto final e a todos os professores do curso de Engenharia da Computação que passaram com paciência e profissionalismo todo o conhecimento, principalmente aos professores Thiago Toribio e Osmar Quirino que me supervisionaram e me orientaram sabiamente durante o desenvolvimento deste projeto.

A todas as pessoas que auxiliaram para que esse projeto fosse concluído com sucesso.

Sumário

Lista de figuras

Lista de quadros

Resumo

Este trabalho tem como objetivo demonstrar a implementação de Algoritmos Genéticos como opção de Inteligência Artificial de forma eficiente e de baixo custo no desenvolvimento de robôs automaticamente móveis em espaços inicialmente desconhecidos com base em dados retirados a partir de um medidor de distância para a entrada de informações do mundo físico. Para isso, é utilizado um protótipo robótico utilizando o kit *Lego Mindstorm NXT* que envia e recebe informações a um computador central que está rodando o Algoritmo Genético via *Bluetooth*. O robô também é capaz de realizar cálculos para a criação de novas rotas podendo se locomover de um ponto inicial para um ponto final (objetivos) – em um determinado espaço já conhecido – sem a necessidade de intervenção humana.

Para apresentação do mesmo em banca avaliadora, são apresentados videos demonstrando sua performace em vários ambientes reais e uma demonstração ao vivo em um ambiente reduzido montado pelo Autor.

Palavras Chave: Algoritmos Evolutivos, Algoritmos Genéticos, Inteligência Artificial, Robótica.

Abstract

This project aims to demonstrate the implementation of Genetic Algorithms as an option on Artificial Intelligence in an efficient mode and low cost in developing automatically mobile robots on initially unknown spaces based on data taken from a distance meter that enter information of the physical world. It is used for prototype a robot using the Lego Mindstorm NXT kit that sends and receives information to a central computer that is running the Genetic Algorithm via Bluetooth. The robot is also able to perform calculations for the creation of new routes being able to move from a starting point for a period (goals) - in a particular area already known - without the need for human intervention. For presentation for the judges, are presented videos showing your perform in various environments and a real live demonstration on a reduced environment fitted by the Author.

Keywords: Evolutionary Algorithms, Genetic Algorithms, Artificial Intelligence, Robotics.

CAPÍTULO 1 - INTRODUÇÃO

Mesmo antes da existência da tecnologia atual, o ser humano vem utilizando de técnicas e mecanismos que possam facilitar seu trabalho, como o emprego de animais – boi, cavalo, cachorro etc – até a criação de ferramentas e máquinas – pá, enxada, foice, carroça etc. Com o passar do tempo, máquinas automatizadas começaram a serem implementadas e, junto com essas, o uso de robôs para aumentar a produtividade e garantir a segurança do ser humano.

O objeto de estudo desse trabalho de pesquisa encontra-se na classe dos robôs exploradores, utilizados para diversas tarefas como a exploração de cavernas, prédios em desabamento e demais ambientes desconhecidos e/ou perigosos para o homem. Tem-se como objetivo o estudo e implementação de Algoritmos Genéticos para o desenvolvimento de um robô automaticamente móvel de baixo custo financeiro em um espaço inicialmente desconhecido com base em dados retirados de periféricos simples para entrada de dados do mundo físico, como sensores, incluindo um medidor de distância dos pontos do espaço real. O robô também deve ser capaz de realizar cálculos para a criação de novas rotas, podendo locomover-se de um ponto inicial para um ponto final (objetivo) sem a intervenção humana.

Robôs exploradores, automáticos ou remotamente controlados, são utilizados tanto para fins militares (FOLHAONLINE, 2010.) como para fins de pesquisa. Um bom exemplo de robôs exploradores para fins de pesquisa são os robôs provenientes do antigo programa "*NASA Space Telerobotics Project*" (NASA, 2010.)

Desta forma, espera-se que este trabalho possa contribuir para a engenharia da computação para futuros estudos na aplicação de Algoritmos Genéticos e demais opções de inteligência artificial na implementação da robótica.

1.1 Motivação

Normalmente o planejamento de trajetória para robôs móveis é feito fundamentado em algoritmos tradicionais de otimização. São também utilizadas estratégias heurísticas, métodos que não seguem um percurso claro, baseando-se na intuição e nas circunstâncias com o

objetivo de gerar um novo conhecimento. Esses métodos, apresentam boas soluções para este problema, mas exigem de conhecimento profundo e sistemático do espaço de busca. O uso de Algoritmos Genéticos dispensa esta necessidade.

Algoritmos Genéticos são algoritmos de busca baseados nos mecanismos de seleção natural e na genética. Estes algoritmos implementam estratégias de busca paralelas e aleatórias para solucionar problemas de otimização.

No entanto, por apresentarem buscas aleatórias não significa que não possuam direção. Eles exploram eficientemente informação histórica para encontrar novos pontos de resultado com expectativa de melhora de performance.

As técnicas dos Algoritmos Genéticos simulam processos de sistemas naturais que são indispensáveis à existência do processo evolutivo utilizando, especialmente, os princípios de Darwin. Na natureza, as características de cada indivíduo determinam a sua capacidade de sobrevivência, onde os indivíduos com maior facilidade de adaptação possuem melhores probabilidades de sobreviverem e transmitirem suas características às próximas gerações. Processos computacionais podem ser modelados de forma semelhante, visando a obtenção de melhores resultados na solução de um problema.

1.2 Objetivos

A partir do uso dos Algoritmos Genéticos, foi possível programar o robô de modo que ele tenha dois objetivos principais, que são escolhidos pelo usuário por meio de um menu de configuração: mapeamento geral, onde o robô deve mapear todo o ambiente em que se situa de modo a criar sua planta digital que posteriormente é apresentada para o usuário; e, busca de caminhos, onde o robô utiliza de objetivos – ida de um ponto inicial a um ponto final - desta vez em um ambiente já conhecido em que se situa, de modo que consiga locomover-se livremente entre, por exemplo, os diversos cômodos de uma casa, evitando o contato com os obstáculos que já foram mapeados.

A parte lógica é dividida em duas partes. A primeira, responsável pela movimentação do robô, é desenvolvida utilizando a linguagem de programação Java com a API LeJOS (*Application Programming Interface* – Interface de Programação de Aplicativos), que roda sobre a plataforma Lego Mindstorm NXT 2.0, do grupo Lego, firmware LeJOS versão 0,85, por possuir aspectos que facilitam a criação mecânica, elétrica e lógica na área da robótica. A

segunda, responsável pelo processamento de dados, é desenvolvida utilizando também a linguagem de programação Java, juntamente com a API LeJOS para computadores, que é responsável pelo tratamento dos dados e utilização do Algoritmo Genético, e a API (*Application Programming Interface* – Interface de Programação de Aplicativos) gráfica Java Swing, responsável pela manipulação e geração de imagens.

1.3 Definição do Problema

O problema a ser estudado e solucionado neste projeto de pesquisa é o de um robô automaticamente móvel em um ambiente qualquer inicialmente desconhecido usando como interface de interação com o mundo físico um medidor de distancia ultra-sônico.

É objeto deste estudo, dentre as diversas classes de robôs encontradas, a classe dos robôs exploradores, que têm como objetivo explorar um determinado ambiente, que não precisa ser, necessariamente uma superfície plana, mas, também, pode ser um determinado espaço ou inclusive um objeto fixo e revelar, por meio de sensores, suas características físicas.

1.3.1 Escopo do problema

O problema deste projeto de pesquisa consiste em demonstrar o uso de algoritmos genéticos no desenvolvimento da robótica de baixo custo financeiro a partir da construção de um robô automaticamente móvel que se movimenta em um cenário terrestre inicialmente desconhecido. A partir daí, o robô é capaz de determinar a distância dos objetos e obstáculos ao seu redor e conseguir se locomover livremente, armazenando um arquivo com informações dos caminhos já conhecidos para que possa tomar decisões no decorrer do processo.

Caso o robô já tenha conhecimento do ambiente onde se situa, ou seja, um mapa a partir de uma base de dados coletada de uma leitura anterior, o usuário tem a opção de programar objetivos – ida de uma coordenada a outra – ao robô, que é capaz de locomover-se em sua rota sem intervenção humana, porém não significando que esse é o caminho mais rápido ou mais curto.

1.3.2 Proposta de solução

Utilizando o kit *Lego Mindstorm NXT* 2.0 para a criação de um robô exploratório móvel genérico, este projeto de pesquisa visa focar a solução do problema, de modo genético por meio da criação de um Algoritmo Genético que pode ser implementado em qualquer robô que preencha os requisitos mínimos de estrutura.

O robô possui um sensor físico, medidor de distância ultra-sônico, utilizado para calcular a distância em linha reta de um ponto a outro. Por limitação do sensor de distância utilizado na medição de objetos (dois metros e meio) e também das funções primitivas utilizadas na linguagem escolhida para implementação do projeto (as funções de controle do sensor ultra-sônico encontradas na API LeJOS - *Application Programming Interface* – Interface de Programação de Aplicativos – só conseguem medir distancias de um modo seguro – ou seja, garantindo o valor retornado – de até um metro e setenta centímetros) e visando a facilitação do desenvolvimento da lógica do algoritmo de determinação da distância entre os pontos, a distância máxima a ser medida será limitada em um metro e meio, obrigando o robô a fazer um recálculo dos seus dados garantindo um possível replanejamento na trajetória.

Com base nos dados coletados através do sensor ultra-sônico, o robô é capaz de realizar reconhecimento de obstáculos e movimentação sem contatá-lo. Dados retirados a partir do contador de passos dos servomotores auxiliam o algoritmo de representação gráfica a definir as distâncias percorridas. Através de uma conexão *bluetooth*, o robô comunica-se com um computador, que é responsável pelo tratamento e processamento dos dados adquiridos pelo robô, como também tem a responsabilidade de realizar e devolver cálculos que indicarão o próximo movimento do veículo, tendo este somente a obrigação de controle dos servomotores e arrecadação de dados a serem processados.

O robô não é capaz de aprender e utilizar conhecimentos de antigas rotas e caminhos no processamento de reconhecimento de novos ambientes. O robô não é capaz de desviar-se de objetos em tempo real, somente a partir de sua análise, o que o obriga a não reconhecer objetos móveis, ou seja, que são capazes de mudar sua posição no mundo físico. O robô não trabalha com rotas em terrenos nivelados, somente processa dados de distância e movimentos nos eixos X e Y de um plano cartesiano. O robô não possui sensores medidores de movimento

– como por exemplo, um giroscópio - o que impossibilita a checagem e verificação da distância real de locomoção, deixando-a por conta dos servomotores e do próprio sensor de distância dos objetos, o que implica que o movimento em terrenos com variados tipos de aderência podem gerar cálculos equivocados ao computador que processa os dados. O robô não contém outros sensores de medição de distância além do sensor ultra-sônico, o que pode acarretar falhas na medição de distâncias em face da não uniformidade das superfícies. O robô não armazena dados em sua memória e por isso perde todos os valores de um ambiente mapeado após o término do projeto, sendo responsabilidade do usuário alimentar o robô com os dados de sua atual posição antes de iniciar o módulo de objetivos de um ambiente já conhecido.

Foram realizados vários testes do robô em ambientes desconhecidos, partindo de diversos pontos de origem diferentes.

1.4 Metodologia

Para a realização da montagem do protótipo, foram utilizadas diversas pesquisas bibliográficas sendo as principais: livros, sites de internet e monografias apresentadas anteriormente. Foram feitos também vários estudos e testes de componentes elétricos, linguagens de programação e softwares para análise dos melhores componentes digitais a serem utilizados na implementação.

Para atingir os objetivos propostos, utilizou-se vários componentes mecânicos, eletrônicos e digitais, sendo o principal o kit de robótica *Lego Mindstorm NXT 2.0*.

1.5 Estrutura da monografia

Esta monografia é composta de 6 capítulos, iniciando com a INTRODUÇÃO, que apresenta a motivação do projeto, os principais objetivos, a metodologia de pesquisa e também toda estrutura deste trabalho.

No capítulo 2 são apresentados os FUNDAMENTOS TEÓRICOS, com descrição dos principais assuntos abordados, fazendo menção aos conceitos de inteligência artificial (IA),

algoritmos evolutivos (AE), algoritmos genéticos (AG) e robótica, tratando estes assuntos de maneira que se adequem ao objetivo final proposto por este trabalho de pesquisa.

No capítulo 3 encontra-se a DESCRIÇÃO DO HARDWARE utilizado em todo o projeto, com detalhamento da especificação dos componentes do kit *Lego Mindstorm NXT 2.0* e a especificação técnica do computador responsável pelo processamento dos dados utilizado no projeto.

No capítulo 4 expõe-se o detalhamento da IMPLEMENTAÇÃO da proposta de solução e apresentação da arquitetura e algoritmos desenvolvidos para a solvência do problema.

No capítulo 5 apresenta-se os TESTES E RESULTADOS da solução proposta para o problema.

O capítulo 6 registra-se a CONCLUSÃO e as recomendações e sugestões para o prosseguimento de trabalhos futuros a serem realizados neste segmento.

CAPÍTULO 2 – FUNDAMENTOS TEÓRICOS

Neste capítulo são apresentados os fundamentos teóricos para a realização deste projeto de pesquisa: conceitos de inteligência artificial, algoritmos evolutivos, algoritmos genéticos e robótica.

2.1 Inteligência Artificial

Inteligência Artificial (IA) pode ser definida como o ramo das ciências da computação que está interessado na automação de comportamentos inteligentes (LUGER, 2002). Porém, esta definição não é suficiente, visto que a inteligência em si não é muito bem definida ou compreendida. Embora a maioria dos seres humanos tenham certeza de que conhecem comportamentos inteligentes quando os vêem, a possibilidade de que alguém se aproxime da definição de inteligência, como se a partir desta fosse possível avaliar um programa de computador supostamente inteligente, é duvidosa, mesmo que esses comportamentos englobassem a vitalidade e a complexidade da mente humana (LUGER, 2002).

Dada a impossibilidade de uma definição formal precisa para IA, visto que para tanto seria necessário definir, primeiramente, a própria inteligência, foram propostas algumas definições operacionais: “uma máquina é inteligente se ela é capaz de solucionar uma classe de problemas que requerem inteligência para serem solucionados por seres humanos”(MCCARTHY, 1956); “Inteligência Artificial é a parte da ciência da computação que compreende o projeto de sistemas computacionais que exibam características associadas, quando presentes no comportamento humano, à inteligência” (BARR e FEIGENBAM, 1981); “Inteligência Artificial é o estudo das faculdades mentais através do uso de modelos computacionais” (CHARNIAK e MCDERMOTT, 1985); “[Automatização de] atividades que associamos ao pensamento humano, atividades como a tomada de decisões, a resolução problemas, o aprendizado [...]” (BELLMAN, 1978); “O estudo das computações que tornam possível perceber, raciocinar e agir.” (WINSTON, 1992); “A arte de criar máquinas que executam funções que exigem inteligência quando executadas por pessoas.”(KURZWEIL, 1990).

“Como o problema de definição de inteligência artificial torna-se um problema de definição de inteligência em si, seria a inteligência uma capacidade única ou apenas um nome para um conjunto de habilidades distintas e independentes? [...] O que é criatividade? O que é intuição?” (LUGER, 2002).

Esta é uma pergunta sem resposta, porém ela e todas suas variações têm ajudado a dar forma aos problemas e soluções que constituem o núcleo da inteligência artificial moderna. De fato, parte do atrativo por trás da inteligência artificial é que a mesma apresenta ferramentas únicas e poderosas que exploram exatamente perguntas desse tipo.

Retornando ao ponto inicial sobre a definição de inteligência artificial, porém examinando as ambições e realizações dos estudiosos, é possível distinguir duas áreas de atuação principais: representação de conhecimento e busca de solução. A representação de conhecimento nada mais é do que a tentativa de capturar em uma linguagem formal, ou seja, uma linguagem possível de manipulações por parte do computador, toda a gama de conhecimento necessário para comportamentos inteligentes. Já a busca de solução é uma técnica solucionadora de problemas que sistematicamente explora um espaço de estados (LUGER, 2002), como é o caso desse projeto de pesquisa. Para exemplo, são citados abaixo alguns campos de aplicações e suas determinadas áreas da IA utilizadas (RUSSEL, 2004):

- Jogos (busca de soluções);
- Reconhecimento de linguagem (representação de conhecimento);
- Modelagem de performance humana (busca de soluções);
- Robótica (busca de soluções);
- Planejamento logístico (busca de soluções);
- Modelagem semântica (representação de conhecimento);

2.1.1 Algoritmos tradicionais de busca e otimização de espaço

De acordo com David E. Goldberg, podem ser identificados três tipos principais de métodos de busca: baseada em cálculo, enumerativa e aleatória (GOLDBERG, 1953).

- Busca baseada em cálculo

Este método pode ser divido em duas subclasses: Direta e Indireta.

Com o método de busca baseada em cálculo indireto procura-se por locais nas extremidades do nosso objetivo resolvendo equações geralmente não lineares, obtidos após levar o gradiente da função objetivo igual a zero. Em outras palavras, a partir de cálculos, é possível identificar os pontos que se encontram nas redondezas do objetivo final, conseguindo a partir dali chegar-se ao resultado.

Com o método de busca baseada em cálculo direto, utiliza-se uma otimização da busca usando a própria função objetivo e movendo-se em direção ao seu gradiente. Ou seja, a partir de cálculos, é possível encontrar o ponto exato do resultado final identificando o ponto mais alto da função objetivo, daí a necessidade do uso do seu gradiente.

A figura 1 ilustra uma função de único pico, um exemplo bastante simples para utilização de métodos de busca e otimização baseada em cálculo. Imaginando que a função objetivo é encontrar o ponto mais alto da função, uma simples busca baseada em cálculo se mostra o método mais rápido e eficiente para o problema.

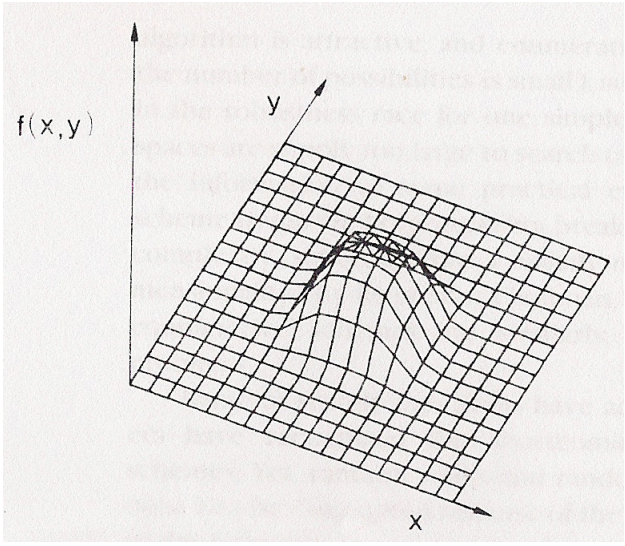

Figura 1 – Função de único pico (GOLDBERG, 1953)

Já a figura 2, apresenta uma figura de múltiplos picos. Claramente, começar o procedimento de busca nas redondezas do pico menor irá fazer com que o algoritmo perca o evento principal, o pico mais alto. Ou seja, após a busca no pico menor, melhorias devem ser feitas a partir de um recálculo em um ponto aleatório ou o uso de algum outro artifício.

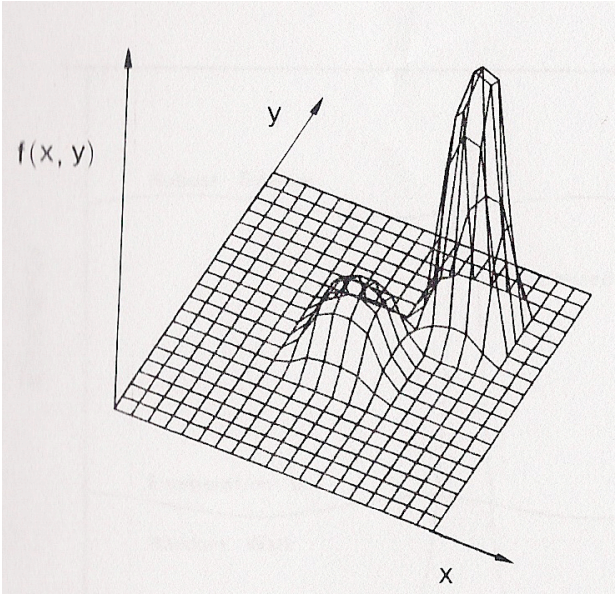

Figura 2 – Função de múltiplos picos (GOLDBERG, 1953)

Contudo, embora este tipo de busca seja o mais apropriado para se encontrar a melhor solução em um espaço de amostragem, ele não se encaixa na proposta do projeto aqui discutido, pois uma busca baseada em cálculos utiliza muitas variáveis, demandando um número muito alto de processamento para efetuar os cálculos necessários e no caso de uma solução genérica como a proposta aqui, torna-se impossível a definição do número de variáveis finais a serem usadas.

- Busca enumerativa

Buscas baseadas em enumerações existem de varias formas e tamanhos. A ideia é muito simples: com um espaço finito de soluções ou ainda um espaço discreto infinito de soluções, o algoritmo de busca começa a verificar os valores de todos os pontos no espaço, um de cada vez, da função objetivo e todas suas possíveis soluções. Porém, apesar da simplicidade desse tipo de algoritmo ser bem atraente, muitos casos podem

ser desconsiderados por uma única razão: eficiência. Muitos espaços de amostragem simples oferecem um espaço de soluções muito grande para serem procurados um a um e ainda serem utilizados em uma solução prática, por isso descartou-se a possibilidade de uso desse tipo de busca nesse projeto. (GOLDBERG, 1953).

Na figura 3, considera-se os pontos de A à K e suas ligações como um determinado espaço de amostragem. Caso o algoritmo queira se movimentar do ponto E ao ponto D, ele terá que processar informação de cada caminho disponível, obtendo finalmente os 17 caminhos disponíveis do nosso ambiente, para depois decidir qual rota tomar, isso sem contar com um algoritmo de melhor caminho disponível, o que ainda obrigaria a um armazenamento de dados em forma de grafo (conjunto de vértices e arestas).

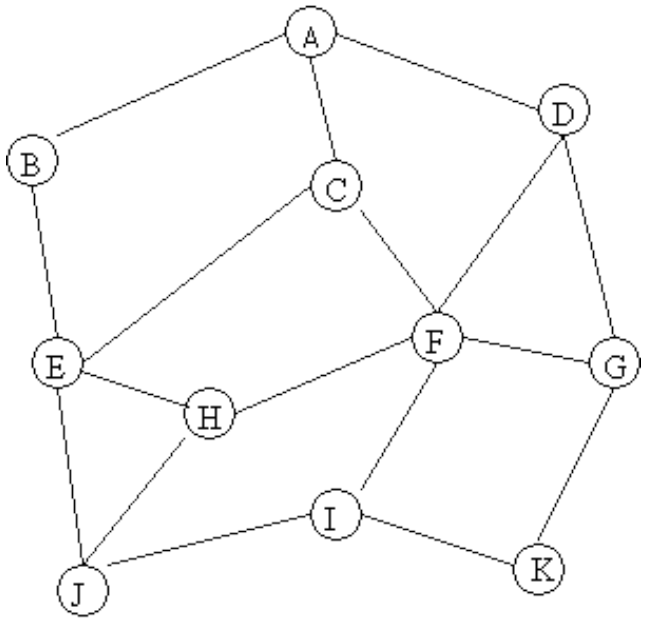

Figura 3 – Grafo (Autor)

- Busca aleatória

Algoritmos de busca aleatórias consistem em procedimentos com resultados estocásticos, ou seja, com resultados aleatórios (como o resultado ao se jogar um dado ou uma moeda, é impossível preceder o valor a ser obtido pois o mesmo depende de um evento aleatório) porém com um direcionamento bem definido, obtido de diversas maneiras sendo a mais comum a criação de uma função objetivo concisa. Apesar de ser

aleatório, um algoritmo de busca aleatória não consiste, necessariamente, em um algoritmo com idas e vindas aleatórias que salva a melhor solução até o momento e compara, devido ao tempo necessário para processar todos esses procedimentos. Com uma busca aleatória realizada deste modo não é possível esperar a longo prazo um resultado melhor do que em uma busca baseada em enumeração. Neste projeto de pesquisa, para evitar o gargalo de perigosos modos de busca aleatórias, deve-se ser cuidadoso para distingui-las de técnicas aleatórias. O algoritmo genético (AG), um das opções de algoritmos evolutivos (AE) e também a escolha de ferramenta deste projeto de pesquisa, é um exemplo de um procedimento de busca que usa a escolha aleatória como ferramenta para guiá-lo para uma busca que explora mais o possível espaço de soluções para um determinado problema. Usar a escolha aleatória como ferramenta em uma busca direta parece estranho à primeira vista, porém a natureza contém vários exemplos para apresentar como casos de sucesso. O mais importante a se tirar daqui é que uma busca aleatória não necessariamente implicará em uma busca sem direção definida. (GOLDBERG, 1953).

Apesar de tudo, deve ser mencionado aqui que os algoritmos evolutivos também apresentam limitações. Os Algoritmos Evolutivos tratam de métodos estocásticos e seu desempenho varia de execução para execução. Devido a isto, a média da convergência sobre diversas execuções do Algoritmo Evolutivo é um indicador de desempenho mais útil que uma simples execução, ou seja, a média global de tempos de processamentos é um fator mais relevante do que o tempo de processamento de uma única execução. Os Algoritmos Evolutivos, nas suas configurações usuais, também apresentam dificuldades para a determinação do ótimo global, sem a utilização de uma metodologia de otimização local, acarretando em outra limitação, a necessidade de análise de todas as amostras do processo a cada avaliação da função de aptidão. Este aspecto é uma limitação relevante para aplicações em tempo real. (GOLDBERG, 1953).

2.1.2 Algoritmos Evolutivos

Algoritmos Evolutivos são aqueles baseados em uma gama de mecanismos da evolução biológica. A motivação para a construção de modelos de soluções computacionais surgiu de teorias através das quais a natureza, por meio de recursos, resolve problemas complexos do

nível de determinação de quantidade de “recursos” para resolver “adversidades”, como por exemplo, de sobrevivência. Assim, pode-se dizer que a natureza otimiza seus mecanismos para resolver um ou mais problemas. (SILVA. L, 2001).

Os principais tipos de Algoritmos Evolutivos são:

- Algoritmos Genéticos (HOLLAND, 1992) – AG;
- Estratégias Evolutivas (RECHENBERG, 1973) - EE;
- Programação Evolutiva (FOGEL, 1966) - PE;
- Programação Genética (KOZA, 1992)- PG;

De acordo com Thomas Baeck (BAECK, 1994), todos esses métodos de algoritmos evolutivos utilizam o mesmo algoritmo básico, só variando na apresentação dos dados, como é descrito no quadro 1, abaixo:

Quadro 1 – Comparativo entre os principais tipos de Algoritmos Evolutivos

	AG	EE	PE	PG
Representação	Cadeias binárias	Vetores reais	Vetores reais	Árvores
Auto-adaptação	Nenhuma	Desvio padrão e co-variâncias	Desvio padrão e coeficiente de co-relação	Nenhuma
A aptidão é	Valor escalonado da função objetivo	Valor da função objetivo	Valor escalonado da função objetivo	Valor escalonado da função objetivo
Mutação	Operador secundário	Principal operador	Único operador	Um dos operadores
Recombinação	Principal operador	Diferentes variações, importante para a auto-adaptação	Nenhuma	Um dos operadores
Seleção	Probabilística	Determinística	Probabilística	Probabilística

(BAECK, 1994)

2.1.2.1 Algoritmos Genéticos

Dentro da classe dos algoritmos evolutivos existem os algoritmos genéticos (AG), que se baseiam nos conceitos da seleção natural e da genética. A partir da combinação da capacidade de sobrevivência dos seres (denominado função de aptidão ou *fitness* pelos pesquisadores da área) com estruturas de dados computacionais, o AG simula criaturas de modo artificial (por meio de strings) que são manipuladas gerando mais e mais seres (com a combinação entre bits e pedaços dos ancestrais mais fortes). (GOLDBERG, 1953).

"O tema central da pesquisa sobre algoritmos genéticos tem sido a robustez, o equilíbrio entre a eficiência e a eficácia necessária para a sobrevivência em muitos ambientes diferentes". (GOLDBERG, 1953).

Os algoritmos genéticos são diferentes dos procedimentos de busca e otimizações normais em quatro pontos (GOLDBERG, 1953):

1. Algoritmos genéticos trabalham com uma codificação do leque de parâmetros, e não com os próprios parâmetros;
2. Algoritmos genéticos buscam uma população de pontos, não um único ponto;
3. Algoritmos genéticos usam a própria função objetivo e não variáveis e outros conhecimentos auxiliadores;
4. Algoritmos genéticos usam regras probabilísticas para transição e não regras deterministas.

Um algoritmo genético simples é constituído de três operações: reprodução, *crossover* e mutação.

Antes de iniciar o procedimento de processamento de dados, o AG é alimentando com as informações a serem trabalhadas e a partir daí é tirado o valor de aptidão de cada uma das possíveis soluções, por meio de uma função objetivo – definida de acordo com a aplicação e resultado esperado. Por exemplo, a função objetivo pode ser uma função quadrática onde procura-se o ponto máximo dela.

- Reprodução

Após a obtenção de todos os valores de aptidão do espaço de amostragem, é possível calcular a porcentagem de incidência de cada valor adquirido.

A verdadeira ação de reprodução dos valores acontece aqui. Com os valores das porcentagens de incidência de cada dado de acordo com seu valor de aptidão monta-se uma roleta imaginária alocando nesta cada valor do espaço de amostragem de acordo com sua porcentagem. Ao se girar a roleta n vezes, sendo n o número de amostras contidos na família inicial de dados, obtém-se uma nova família de valores, sendo que aqueles que obtinham uma aptidão mais elevada sobrevivem, enquanto que os que tiverem a menor aptidão são descartados da nova família. (GOLDBERG, 1953).

- *Crossover*

O processo de *crossover* em um Algoritmo Genético é uma operação que se assemelha ao processo de *crossover* durante a geração de um ser humano no útero de sua mãe. Um ser humano em formação recebe características do pai e da mãe que devem ser encaixadas dentro de seus 46 cromossomos. Logo, parte dos cromossomos é herdada do pai da criança e outra parte da mãe. A operação de *crossover* de um algoritmo genético utiliza esse fato do mundo genético para criar uma nova família de valores no espaço de amostragem. Utilizando um único operador K (que tem uma faixa permitida de valores de 1 até L, sendo L o número de bits de cada string - ou indivíduo, em comparação com a realidade - do espaço de amostragem) são escolhidos randomicamente pares dos novos valores de trabalho, obtidos após a reprodução, para fazerem o *crossover* entre si. Gerando um número randômico para K, que se encontre dentro da sua faixa permitida de valores, trabalha-se em cima do casal de strings obtidos randomicamente da seguinte forma: todos os bits da posição K até a posição final são trocadas entre si, formando agora novos valores de trabalho. O *crossover* deve ser efetuado em todo o espaço de amostragem obtido após a reprodução não permitindo que o mesmo valor faça parte de mais de uma operação de *crossover*. Em uma família de números ímpares de resultados, tem-se um valor que não participa da operação de *crossover*. (GOLDBERG, 1953).

- Mutação

Toma-se um valor muito pequeno a ser usado como a probabilidade de mutação de um bit de um dos valores do espaço de amostragem, como por exemplo o valor 0,001. Aplica-se essa operação em cada bit de cada valor do espaço de amostragem da família gerada após o *crossover*, sendo que cada bit tem a probabilidade de 0,001 de ter o valor invertido (0 ser transformado em 1 e vice-versa). Por exemplo, considera-se uma família de 1000 strings de 7 bits cada, em um espaço de amostragem de um certo problema, ou seja, 7000 bits. Espera-se que dentro de toda essa família, somente 7 bits sejam invertidos. (GOLDBERG, 1953).

Após estas 3 operações serem realizadas, tem-se novos valores no espaço de amostragem a ser analisado, com um novo valor total, médio e máximo de aptidão e com novos valores de porcentagem de sobrevivência de cada amostra.

Esses fundamentos são baseados na construção do algoritmo genético simples. A partir de manipulações do modelo genérico, foram desenvolvidos outros tipos para problemas mais específicos, os quais não são citados aqui por não fazerem parte do intuito deste projeto de pesquisa.

Dentre as variações de Algoritmos Genéticos, este projeto adota o modelo de Algoritmo Híbrido. Utilizando o Algoritmo Genético como uma ferramenta de busca e otimização de uma população de soluções no Algoritmo Híbrido, a mistura das técnicas criam em conjunto uma espécie de aprendizado à lógica final, com módulos de armazenamento e comparação de registros de buscas e otimizações passadas. (HOLLAND, 1992).

2.2 Robótica

A robótica é um ramo recente da tecnologia, tendo início no século XX, que normalmente une mecânica à eletrônica e computação. O produto final da robótica são os robôs, agentes físicos que executam tarefas manipulando o mundo físico. A robótica atual emprega um conjunto diversificado de sensores, inclusive câmeras e ultra-som para medir o ambiente, além de giroscópios e acelerômetros para medir o movimento do próprio robô.

A estrutura básica de um robô é definida pela junção de sensores, que interagem com o mundo físico traduzindo as informações para serem trabalhadas digitalmente; um mecanismo atuador, que fornecerá movimentações ao robô; um mecanismo manipulador, que é a interface de contato do robô com o mundo físico; e uma fonte de energia, sendo que nenhum desses, com exceção da fonte de energia, é uma característica obrigatória de todos os robôs.

A maior parte dos robôs atuais enquadra-se em uma dentre três categorias principais: os manipuladores (ou braços robôs), robôs móveis e, por último, robôs híbridos (robôs móveis equipados com manipuladores). Esse projeto se concentra na categoria dos robôs móveis.

Os robôs móveis deslocam-se por seu ambiente usando rodas, pernas ou mecanismos semelhantes. Eles são projetados para uso na entrega de alimentos em hospitais, para mover contêineres em docas de carga e tarefas semelhantes. Como exemplos de robôs móveis podemos citar: o veículo terrestre não-tripulado (ULV – *unmanned land vehicle*) chamado NAVLAB, capaz de realizar a navegação autônoma sem condutor em auto-estradas; os veículos aéreos não-tripulados (UAV – *unmanned air vehicle*), comumente usados para vigilância, pulverização de lavouras e operações militares; os veículos autônomos subaquáticos (AUV – *autonomous underwater vehicle*), usados em exploração no fundo do mar e os viajantes planetários, como o robô *Sojourner* da NASA, que explorou a superfície de Marte em julho de 1997.

O termo robótica foi criado pelo escritor de Ficção Científica Isaac Asimov no seu romante "*I, Robot*" (Eu, Robô), de 1948 (ASIMOV, 1969). Nessa obra literária, Isaac Asimov criou as famosas três leis da robótica que, segundo ele, regeriam os robôs no futuro:

1. Um robô não pode fazer mal a um ser humano e nem, por omissão, permitir que algum mal lhe aconteça;
2. Um robô deve obedecer às ordens dos seres humanos, exceto quando estas contrariarem a Primeira lei.
3. Um robô deve proteger a sua integridade física desde que, com isto, não contrarie a Primeira e Segunda leis.

Apesar dessas leis não se aplicarem aos robôs atuais por eles executarem aquilo para que foram programados, geralmente tarefas simples e únicas, o autor deste projeto possui como ponto de vista a crença que em um futuro não tão distante, quando a Inteligência Artificial chegar à um patamar onde consiga tomar decisões próprias acerca do mundo ao seu redor, essas leis poderão, e deverão, ser aplicadas como medidas de proteção à vida humana e ao robô em si.

CAPÍTULO 3 – DESCRIÇÃO DE HARDWARE

O hardware utilizado para a construção do presente projeto é composto por:

- Kit de robótica *Lego Mindstorm NXT 2.0*;
- *MacBook Pro* modelo *MacBookPro5,5*.

3.1 Kit Lego Mindstorm NXT 2.0

O kit *Lego Mindstorm* é uma linha de brinquedos da *Lego Groups*, lançada inicialmente em 1998, fruto de uma parceria com o *MIT* (*Massachusetts Institute of Technology*), voltada para educação tecnológica. Em 2006, a *Lego Groups* lançou comercialmente o kit *Lego Mindstorm NXT,* com *hardwares* e peças mais intuitivas e até uma linguagem de programação gráfica, permitindo que até crianças e jovens conseguissem criar um robô em 30 minutos (LEGO, 2010).

Em 2009, a *Lego Groups* lança comercialmente o Kit *Lego Mindstorm NXT* 2.0, que é o utilizado nesse projeto de pesquisa. O kit *NXT* 2.0 possui uma maior e revisada variedade de 619 elementos. O novo conjunto de *hardware* que acompanha o kit consiste de um micro-controlador *NXT*, 3 servo-motores interativos com sensores de rotação acoplados, 1 sensor ultra-sônico, 2 sensores de toque e um sensor de cores com funcionalidade tripla: Sensor de cores – detecta até 6 cores diferentes; Sensor luminoso – detecta diferentes níveis de intensidade luminosa; Lâmpada luminosa – trabalhando com as cores básicas do sistema *RGB* (*Red, Green and Blue* - Vermelho, Verde e Azul) (LEGO, 2010).

3.1.1 Micro-controlador NXT

O micro-cotrolador *NXT* que acompanha o kit possui dois microprocessadores: um processador principal Atmel 32-bit ARM, e 256 KB de memória de armazenamento FLASH, 64 KB de RAM, trabalhando à uma freqüência de 48 MHz; e um co-processadorr Atmel 8-bit

AVR, com 4 KB de memória de armazenamento FLASH, 512 Bytes de RAM, trabalhando à uma freqüência de 8 MHz. (LEGO, 2010).

Para o controle de outros dispositivos, ele possui 3 portas de saída de dados com interface de 6 fios que suporta entrada de dados de codificadores, que são usualmente utilizadas para controle dos servomotores, e 4 portas de entrada de dados com interface de 6 fios, que suportam tanto sinais digitais quanto analógicos, que são utilizadas normalmente para gestão dos sensores do kit. (LEGO, 2010).

Para apresentação de informações, o micro-controlador NXT possui um *LCD* (*Liquid Crystal Display* – Tela de cristal liquido) com suporte gráfico de tamanho 100 x 64 pixels em preto e branco, como apresentado na figura 4. Sua área de visão é de 26 x 40.6 mm. Além da tela é possível ter informações do programa executado no micro-controlador através de saída de informações de um alto falante, com um raio de freqüências suportadas de 2-16 KHz. (LEGO, 2010).

Figura 4 – Microcontrolaor *NXT* (PHILO, disponível em <http://www.philohome.com/nxtmotor/nxtmotor.htm >, Acesso em 18 de Abril de 2010)

Para se comunicar com outros dispositivos, tanto para recebimento de dados – como o recebimento de um código compilado a ser executado – quanto para entrega de informações – como os valores de retorno dos sensores a ser passado para que outro dispositivo possa trabalhar em cima – o micro-controlador NXT possui uma porta *USB* (*Universal Serial Bus* – Barramento Serial Universal) 2.0 com velocidade máxima de 12 Mbits/s e comunicação sem

fio via *Bluetooth* versão 2.0, com memória interna de 47 KByte de RAM e externa de 8 Mbit de memória FLASH, trabalhando com um microprocessador de 26 MHz. (LEGO, 2010).

A alimentação do micro-controlador NXT é feita a partir de 6 pilhas de tamanho AA. É recomendável o uso de pilhas alcalinas, já que essas fornecem 1,5V de tensão e alguns sensores, como o ultra-sônico, necessitam de 9V de alimentação. Porém é possível adaptar o micro-controlador para trabalhar com pilhas recarregáveis de íons de lítio, que normalmente fornecem 1,2V de tensão, sem perdas significativas nos resultados, todavia sem garantias do mesmo. (LEGO, 2010).

3.1.2 Servomotores

O kit *Lego Mindstorm NXT* 2.0 contém 3 servomotores interativos, como o apresentado na figura 5, com sensores de rotação inclusos, apresentado na figura 6, o que permite uma precisão de até 1 (um) grau de rotação em cada movimento.

Figura 5 – Servomotor do kit *Lego Mindstorm NXT* (PHILO, disponível em <http://www.philohome.com/nxtmotor/nxtmotor.htm >, Acesso em 18 de Abril de 2010)

Figura 6 – Motor e parte de trás da placa de cirtuico integrado (PHILO, disponível em <http://www.philohome.com/nxtmotor/nxtmotor.htm >, Acesso em 18 de Abril de 2010)

A relação de engrenagens do servomotor do kit é de 1:48, como é apresentado na figuras 7 e 8.

Figura 7 – Engrenagens do codificador e bifurcação ótica que fornece funções de sensor de rotação (PHILO, disponível em <http://www.philohome.com/nxtmotor/nxtmotor.htm >, Acesso em 18 de Abril de 2010)

O pesquisador Philippe E. Hurbain fez vários testes de performance com o servomotor NXT utilizando alimentação 9V (pilhas alcalinas, de 1.5V cada) e alimentação 7.2V (pilhas de íons de lítio, de 1.2V cada) e divulgou os resultados em seu site oficial (PHILO, 2010).

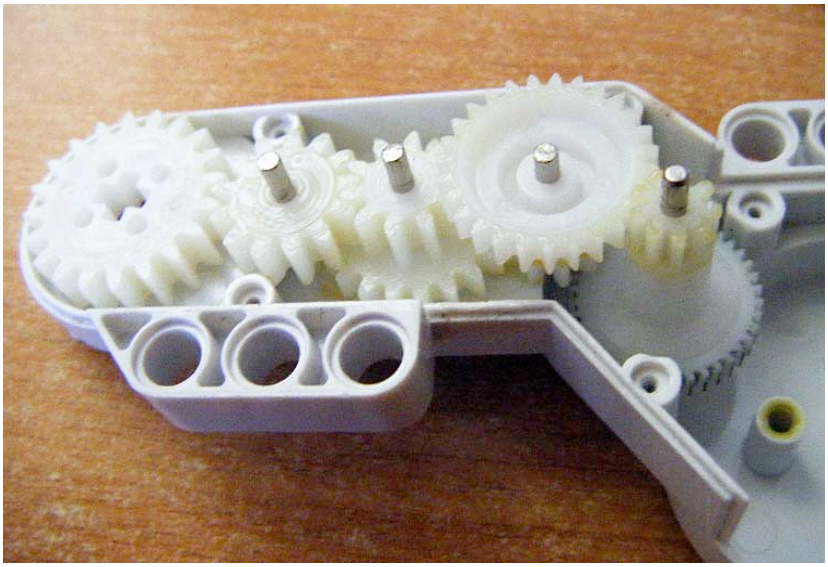

Figura 8 – Engrenagens do motor (PHILO, disponível em <http://www.philohome.com/nxtmotor/nxtmotor.htm >, Acesso em 18 de Abril de 2010)

3.1.3 Sensores

O kit *Lego Mindstorm NXT* 2.0 vem acompanhado de 4 sensores, de 3 tipos. 2 sensores de toque, 1 sensor luminoso e 1 senso ultra-sônico.

3.1.3.1 Sensor de toque

A figura 9 apresenta o sensor de toque, que reage à pressão e liberação, de uma forma booleana. Ele pode detectar toques simples ou múltiplos (configurados via algoritmo) e enviar respostas para o micro-controlador NXT. Além do mais ele vem com uma interface na área de toque que permite o acoplamento de outras peças, abrindo portas para a criação de estruturas sensíveis a toque.

Figura 9 – Sensor de toque o kit Lego Mindstorm NXT (LEGO, Mindstorm NXT. Disponível em <http://mindstorms.lego.com/en-us/products/default.aspx#9843>. Acesso em 6 de Junho de 2010)

3.1.3.2 Sensor luminoso

O sensor luminoso é um dos possibilitam a capacidade de visão para os robôs. Atualmente, o sensor luminoso possui 3 funções em 1. Ele pode detectar 6 cores diferentes, ler a intensidade da luz em um cômodo e medir a intensidade de luz de superfícies coloridas. Além disso, pode ser usado como uma lanterna de cores.

3.1.3.3 Sensor Ultra-sônico

O sensor ultra-sônico é o outro sensor que dá a habilidade de visão para os robôs, e o mesmo se assemelha com um par de olhos – figura 10. Pode ser usado para medir distâncias, desviar de objetos automaticamente e também detectar movimento.

Figura 10 – Sensor ultra-sônico do kit *Lego Mindstorm NXT* (LEGO, Mindstorm NXT. Disponível em < http://mindstorms.lego.com/en-us/products/9846.aspx>. Acesso em 6 de Junho de 2010)

O sensor ultra-sônico mede distâncias já convertendo o valor para centímetros ou polegadas diretamente para o micro-controlador NXT. Ele é capaz de medir distâncias a um ponto máximo de 255 cm, com uma precisão de 3 cm.

O sensor ultra-sônico consegue medir a distância dele até objetos a partir da emissão de pulsos de ultra-som (som com uma freqüência superior ao que ouvido humano consegue perceber). A superfície plana do alvo reflete o ultra-som que retorna ao sensor como um eco. A partir do tempo da emissão até o retorno do ultra-som são feitos cálculos, levando em consideração a velocidade do som no ar para se ter como resultado a distância entre o sensor e um determinado objeto.

Ademais, o sensor ultra-sônico é o único sensor digital que acompanha o kit Lego Mindstorm NXT 2.0 que tem como conseqüência a necessidade de alimentação de 9V para funcionar com total êxito. (LEGO, 2010).

3.1.3 Barramento

O kit *Lego Mindstorm NXT* 2.0 vem acompanhado de 7 cabos – 1 de 20cm, 4 de 35cm e 2 de 50cm. O barramento utilizado pelo micro-controlador *NXT* para se conectar aos seus sensores e motores são compostos de 6 fios em uma interface com conectores RJ12 estilizada pelo LEGO Groups, apresentados com detalhes nas figuras 11 e 12. (PHILO, 2010).

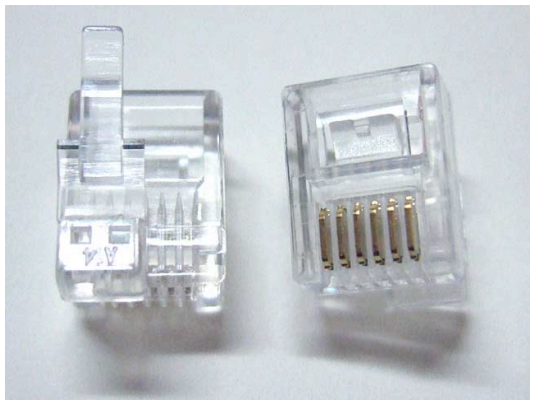

Figura 11 – Conector RJ42 customizado para o uso no micro-controlador NXT. (PHILO, Disponível em <http://www.philohome.com/nxtplug/nxtplug.htm>. Acesso em 6 de Junho de 2010)

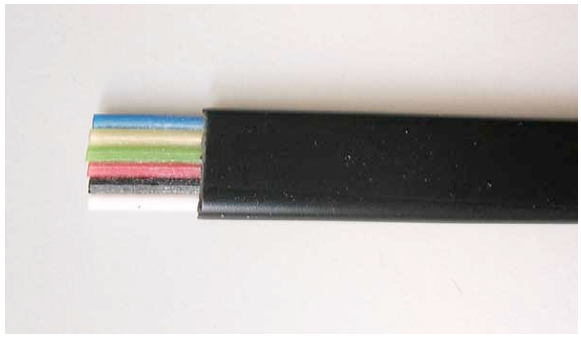

Figura 12 – Fiação dos cabos conectores do kit *Lego Mindstorm NXT*. (PHILO, Disponível em <http://www.philohome.com/crimp/crimp.htm>. Acesso em 19 de Maio de 2010)

Quadro 2 – Função de cada um dos fios dos conectores do kit *Lego Mindstrom NXT*

Pino	Cor	Função
1	Branco	Interface Analógica. Para sensores digitais, o pino 1 pode ser configurado para fornecer alimentação de 9V constante
2	Preto	Fio terra
3	Vermelho	Fio terra
4	Verde	Alimentação constante de 4.3V
5	Amarelo	Entrada e saída de dados digitais
6	Azul	Entrada e saída de dados digitais

(NXTASY, 2010)

Para conexão digital entre o micro-controlador NXT e os sensores digitais, a *LEGO Groups* decidiu optar pelo protocolo I^2C. O I^2C é uma interface de comunicação bidirecional que utiliza dois fios desenvolvida pela *Philips Semicondutors* durante a década de 1980. Basicamente, a comunicação digital do micro-controlador NXT utiliza duas linhas: o pino 5 controla o *clock*, e o pino 6 é responsável por transmitir informação bidirecionalmente entre o micro-controlador NXT e o dispositivo conectado a ele – sempre com o NXT agindo como dispositivo mestre, requisitando a comunicação). (PHILO, 2010).

A linguagem de programação NXT-G, que acompanha o software do kit *Lego Mindstorm NXT* 2.0, não explora o potencial completo dos sensores que utilizam o protocolo I^2C (no caso deste projeto de pesquisa, o sensor ultra-sônico), pois dificultaria a programação, que não é o intuito da linguagem.

3.1.4 Bluetooth

"A tecnologia sem fio Bluetooth é o padrão sem fio mundial para conexão pessoal apropriada para uma ampla gama de dispositivos eletrônicos – de telefones móveis e fones para carros, tocadores MP3, câmeras e impressoras." (BLUETOOTH SIG, 2010).

O Bluetooth é uma especificação para WPANs (Wireless personal area networks – áreas de redes pessoais sem fio). Utilizando a tecnologia Bluetooth, é possível trocar informações entre diversos dispositivos como telefones celulares, computadores, impressoras, câmeras digitais entre outros, através de uma freqüência de rádio de curto alcance entre as faixas de freqüência de 2400MHz à 2483,5MHz.(BLUETOOTH SIG, 2010).

O micro-controlador NXT oferece suporte a comunicação sem fio utilizando tecnologia *Bluetooth* com a inclusão de um chip *CSR BlueCore 4 version 2*. O micro-controlador NXT pode se conectar sem a necessidade de fios com outros três dispositivos ao mesmo tempo, porém se comunicando com um de cada vez, através do SSP (*Serial Port Profile* – Perfil de Porta Serial), o que pode ser considerado uma porta serial sem fio. (LEGO, 2010).

A decisão de utilização do *Bluetooth* de classe 2 por parte do *Lego Groups* foi tomada visando a economia de bateria, o que significa que o micro-controlador NXT só pode se comunicar a uma distância máxima de aproximadamente 10 metros. (LEGO, 2010).

Todas as funções que utilizam o *Bluetooth* no micro-controlador NXT utilizam o canal de comunicação *master/slave* (mestre/escravo). Isso significa que um micro-controlador NXT na rede precisa funcionar como mestre para que outro micro-controlador NXT possa enviar mensagem para ele, comunicar-se, como apresentado a figura 13. De acordo com o *firmware* padrão do kit *Lego Mindstorm NXT*, um micro-controlador NXT não é capaz de funcionar como mestre e escravo ao mesmo tempo enquanto se comunica com outros micro-controladores NXT. (LEGO, 2010).

A conexão *Bluetooth* com outros dispositivos, como um computador, ocorre através de canais de comunicação. O micro-controlador NXT possui quatro canais de comunicação utilizados para comunicação via *Bluetooth*. O canal 0 é sempre usado pelo micro-controlador NXT em estado escravo em comunicação com um dispositivo em estados mestre, enquanto os canais 1, 2 e 3 são sempre utilizados para envio de dados de um dispositivo mestre para um dispositivo escravo. (LEGO, 2010).

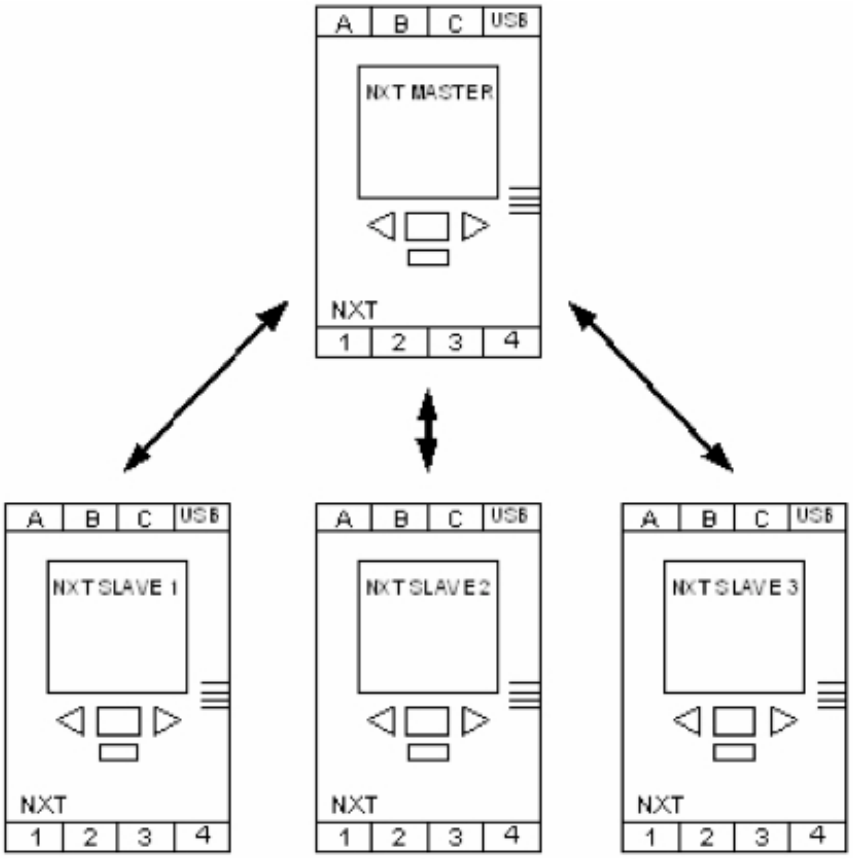

Figura 13 – Modelo de conexão entre micro-controladores NXT (LEGO. Bluetooth Developer Kit. Disponível em <http://mindstorms.lego.com/en-us/support/files/default.aspx> Acesso em 19 de maio de 2010.)

3.1.4.1 Comunicação entre dispositivos Bluetooth e o micro-controlador

O micro-controlador NXT pode comunicar-se com dispositivos *Bluetooth* externos que utilizem o SSP (*Serial Port Profile* – Perfil Porta Serial) e possam ser programados para utilizar o protocolo de comunicação do kit *Lego Mindstorm NXT*. (LEGO, 2010).

Também é possível enviar comandos diretos para o micro-controlador NXT. Comandos diretos são interpretados pelo micro-controlador NXT e traduzidos em funções específicas sem adicional interação do usuário. Isso permite o controle direto de um micro-controlador NXT a partir de um dispositivo Bluetooth externo, como um celular ou um computador. (LEGO, 2010).

3.3 Computador (Processamento Remoto)

Para o processamento de dados e tomada de decisões, é utilizado um *MacBook Pro* (modelo *MacBookPro5,5* apresentado na figura 14) que possui um processador *Intel Core 2 Duo* de 2.26 GHz (modelo P8400), com 2 núcleos de processamento independentes em uma única pastilha de circuito, 3MB de cache level 2, barramento de 1066 MHz, 2 GB de RAM DDR3, 160 GB de HD Serial ATA (5400 RPM), placa gráfica NVIDIA GeForce 9400M com 256 MB de DDR3 e uma tela *widescreen* de 13.3 polegadas (resolução nativa de 1280x800 pixels). Para conexão remota o computador possui *Bluetooth* versão 2.1+EDR, duas portas USB 2.0 e outras conexões que não são utilizadas nesse projeto. (EVERYMAC, 2010)

Figura 14 – Modelo de conexão entre micro-controladores NXT (LEGO. Bluetooth Developer Kit. Disponível em <http://mindstorms.lego.com/en-us/support/files/default.aspx> Acesso em 19 de maio de 2010.)

CAPÍTULO 4 – IMPLEMENTAÇÃO

4.1 Arquitetura do Robô

O robô possui seu eixo principal em cima de dois motores, posicionados paralelamente. Cada motor é capaz de controlar uma esteira tornando possível a movimentação em linha reta, em curva e rotação em cima do próprio eixo.

Em cima do robô é acoplado outro motor com o sensor medidor de distância ultra-sônico, de modo que o robô tenha conhecimento dos objetos à sua volta sem precisar mover-se, rotacionando somente o motor superior.

4.2 Algoritmos

Para o controle total do projeto, foram construídos três algoritmos principais distribuídos entre classes Java:

- Algoritmo de controle do robô (somente 1 classe);
- Algoritmo de processamento lógico (4 classes);
- Algoritmo de processamento gráfico (1 classes).

4.2.1 Algoritmo de controle do robô

O algoritmo de controle do robô é um receptor e interpretador de comandos, estes decididos pelo algoritmo de processamento lógico.

Como a figura 15 demonstra, o algoritmo de controle do robô utiliza 3 classes da API (*Application Programming Interface* – Interface de Programação de Aplicativos) Java:

- java.io.DataInputStream: Permite que a aplicação consiga ler tipos de dados Java primitivos a partir de um canal de entrada de dados;
- java.io.DataOutputStream: Permite que a aplicação consiga escrever tipos de dados Java primitivos em um canal de saída de dados;

• java.io.IOException: Captura exceções nos canais de entrada e saída de dados, relatando os erros ao usuário.

Ainda a partir da figura 15, além das classes da API Java, o algoritmo também importa 7 classes da biblioteca LeJOS (Java for Lego Mindstorm):

• lejos.nxt.Motor: Implementa funções dos servomotores;

• lejos.nxt.NXT: Implementa funções básicas do micro-controlador NXT;

• lejos.nxt.SensorPort: Implementa funções das portas de entrada de dados do micro-controlador NXT;

• lejos.nxt.UltrasonicSensor: Implementa as funções do sensor de distância ultra-sônico;

• lejos.nxt.comm.BTConnection: Fornece implementação para a abertura de um canal de comunicação via Bluetooth;

• lejos.nxt.comm.Bluetooth: Implementa conexão Bluetooth;

• lejos.robotics.navigation.SimpleNavigator: Fornece funções de suporte e navegação de robôs móveis sobre dois servomotores.

```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;

import lejos.nxt.Motor;
import lejos.nxt.NXT;
import lejos.nxt.SensorPort;
import lejos.nxt.UltrasonicSensor;
import lejos.nxt.comm.BTConnection;
import lejos.nxt.comm.Bluetooth;
import lejos.robotics.navigation.SimpleNavigator;
```

Figura 15 – Bibliotecas importadas pelo algoritmo de controle do robô (Autor)

A figura 16 apresenta o processo de criação de constantes e objetos que compõem o algoritmo de controle do robô. No início do algoritmo são declaradas duas constantes:

• diferencaAnguloMedicao: A diferença, em graus, do ângulo de uma medição de distância a outra

• numeroAngulosMedidos: A partir da constante diferencaAnguloMedicao, é calculado o número de medições que serão feitas.

Logo após, é declarado um vetor que responsável por armazenar as distâncias medidas antes de serem enviadas ao computador para processamento. O sensor ultra-sônico é configurado para trabalhar na porta de entrada de dados número 1 na configuração *ping* (ou seja, a leitura de distância será feita aos poucos, e não continuamente. Isso colabora para um maior controle na leitura de distância entre o robô e obstáculos a sua volta). No próximo comando, o micro-controlador NXT é programado para esperar por uma conexão Bluetooth vinda de qualquer lugar. Quando essa conexão é efetuada, são abertos dois canais de entrada e saída de dados via Bluetooth: dis e dos. As duas últimas linhas apresentadas na figura 16 são responsáveis por alimentaram o *software* com as informações físicas do robô (a criação de um novo objeto *SimpleNavigator* utiliza o diâmetro da roda e o tamanho da faixa do robô, além de determinar qual servomotor está no lado direito e no esquerdo do robô, permitindo o controle independente dos dois).

Após efetuadas todas as configurações iniciais do robô, este parte para um ciclo contínuo de espera de comando do computador. As principais funções do robô são as de leitura de distâncias ao seu redor e movimentação.

```
public static void main(String[] args) throws IOException, Exception {
    final int diferencaAnguloMedicao = 10;
    final int numeroAngulosMedidos = 360/diferencaAnguloMedicao;
    int [] distanciasFinal = new int [numeroAngulosMedidos];

    UltrasonicSensor ultrasonico = new UltrasonicSensor(SensorPort.S1);
    ultrasonico.ping();

    BTConnection btc = Bluetooth.waitForConnection();

    DataInputStream dis = btc.openDataInputStream();
    DataOutputStream dos = btc.openDataOutputStream();

    SimpleNavigator robo = new SimpleNavigator(32.2f,186f, Motor.B, Motor.C);
    robo.setPosition(0,0,0);
```

Figura 16 – Declaração de objetos e definição de dados do algoritmo de controle do robô (Autor)

O código de leitura e envio de distâncias começa ajustando a velocidade do servomotor que rotaciona o sensor ultra-sônico. Na figura 17, essa velocidade é igual à diferença de ângulos entre uma medição e outra, ou seja, 10 graus por segundo (essa velocidade pequena

foi tomada devido ao fato dos servomotores do micro-controlador NXT não conterem freios. Seus passos são medidos a partir de um cálculo de tempo e inércia dos motores e então o micro-controlador liga e desliga o servomotor automaticamente. Uma velocidade pequena também diminui a inércia do motor, o que resulta em menos erros de aproximações). Um detalhe importante a se ressaltar é a espera de 300 microssegundos (método Thread.sleep(300)) entre as leituras de distância do sensor ultra-sônico. A API LeJOS recomenda que espere no mínimo 20 microsegundos para o recebimento correto da distância dos objetos ao sensor ultra-sônico. O projeto utiliza 300 microsegundos por questão de segurança e por essa diferença ser quase imperceptível pelos seres humanos. Essa espera existe para que a onda ultra-sônica enviada pelo sensor seja capaz de refletir em algum objeto e retornar ao sensor. Caso esse tempo não existisse, erros de medição poderiam ocorrer com freqüência. Após todas as leituras, a velocidade do motor é ajustada para 900 graus por segundo para que retorne de uma vez à sua posição inicial. O robô então envia as distâncias para o computador a partir de suas requisições.

```
case 5: //Leitura de distâncias
        Motor.A.setSpeed(diferencaAnguloMedicao);
        {
            int i;
            for (i=0;i<distanciasFinal.length;i++)
                distanciasFinal[i]=-20;
            for (i=0;i<distanciasFinal.length;i++){
                Motor.A.rotate(diferencaAnguloMedicao);
                while(Motor.A.isMoving());
                    distanciasFinal[i]=ultrasonico.getDistance();
                    Thread.sleep(300);
                }
            Motor.A.setSpeed(900);
            Motor.A.rotate(diferencaAnguloMedicao*distanciasFinal.length*(-1));
        }
        {
            int i;
            i = dis.readInt();
            while(i<numeroAngulosMedidos){
                dos.writeInt(distanciasFinal[i]*10);
                dos.flush();
                i = dis.readInt();
            }
        }
        break;
```

Figura 17 – Algoritmo de controle do robô. Código de leitura e envio de distancias. (Autor)

Apesar do algoritmo de controle de robô apresentar códigos independentes de movimentação e rotação de robô, o algoritmo de processamento lógico (no qual se encontra o Algoritmo Genético) utiliza de coordenadas cartesianas para locomover e reconhecer a posição exata do robô no mundo real. Como o robô não possui sensores de posicionamento (como giroscópio e acelerômetro), todos os cálculos são feitos a partir dos sensores de rotação embutidos nos servomotores e cálculos feitos pelo objeto *SimpleNavigator* da API LeJOS que utilizam o diâmetro das rodas do robô e o tamanho da sua faixa. Com a velocidade de cada motor ajustada para 720 graus por segundo (ou seja, 2 voltas completas por segundo), o método robo.goTO(x,y), encontrado com detalhes na figura 18, calcula automaticamente o ângulo e a distância a ser efetuada pelo robô (a movimentação do robô foi ajustada para que ele sempre ande em linha reta. Quando ele precisa mudar o ângulo de sua rota, os servomotores rotacionam-se cada um em direção inversa em relação ao outro, fazendo com que o robô gire em volta do próprio eixo).

```
case 3: //Ir para posição X Y
        {
            Motor.B.setSpeed(720);
            Motor.C.setSpeed(720);
            int x,y;
            x=dis.readInt();
            y=dis.readInt();
            robo.goTo(x,y);
            while(Motor.B.isMoving() || Motor.C.isMoving());
        }
        break;
```

Figura 18 – Algoritmo de controle de robô. Código de movimentação em coordenadas cartesianas. (Autor)

4.2.2 Algoritmo de processamento lógico

O algoritmo de processamento lógico está divido em 4 classes:

• PCtoNXT.java: É a classe principal do algoritmo. É responsável por intermediar a comunicação entre o Algoritmo Genético e Robô;

• DispositivoBluetooth.java: Responsável por configurar a conexão Bluetooth com o micro-controlador NXT e enviar e receber dados;

• LogPCtoNXT.java: Responsável por gerenciamento dos arquivos de registro das leituras e dados processados pelo Algoritmo Genético. Também é encarregado de carregar os dados dos registros para geração de gráficos pelo algoritmo de processamento gráfico;

• AlgoritmoGeneticotoNXT.java: É a classe principal do projeto. Responsável por processar os dados e a partir daí decidir as próximas ações do robô. Controla as informações de entrada e saída de registros assim como quando o programa deve ser finalizado automaticamente.

Juntas, essas quatro classes formam o Algoritmo de processamento de dados e decisão de ações do robô.

4.2.2.1 PCtoNXT.java

Como apresentado na figura 19, a classe PCtoNXT.java só utiliza duas classes da API Java:

• java.io.IOException: Para controle de erros durante a comunicação de entrada e saída de dados do Bluetooth;

• java.util.Scanner: Controladora de entrada de dados do teclado, utilizada para inserção do nome do projeto a ser criado ou carregado.

```
import java.io.IOException;
import java.util.Scanner;
```

Figura 19 – Bibliotecas importadas pela classe PCtoNXT.java (Autor)

A figura 20 apresenta a declaração de constante, algumas variáveis de trabalho e objetos instanciados pela classe “PCtoNXT.java”. As variáveis “diferencaAnguloMedicao” e “numeroAngulosLidosRobo” são constantes declaradas em todas as classes do projetocom a mesma função: controlar o numero de medições feitas pelo sensor ultra-sônico do robô. Mais além, são declaradas 4 variáveis de controle de dados enviados ao robô, porém somente duas delas são realmente usadas no estágio final do projeto: “proximoX” e “proximoY”, que contém os dados das coordenadas do próximo ponto a ser analisado pelo robô. A variável

“nomeProjeto” é auto-explicativa, sua função é armazenar o nome do projeto a ser iniciado ou carregado.

Na figura 20, ainda é possível notar que a classe “PCtoNXT.java” cria instâncias das classes “DispositivoBluetooth.java” e “AlgoritmoGeneticotoNXT”, deixando claro que é a intermediadora entre estas duas últimas.

```
static final int diferencaAnguloMedicao = 10;
static final int numeroAngulosLidosRobo = 360/diferencaAnguloMedicao;
static private int distancia,angulo,proximoX,proximoY;
static private String nomeProjeto;
static DispositivoBluetooth dispositivoBluetooth = new DispositivoBluetooth();
static AlgoritmoGeneticotoNXT algoritmoGenetico = new AlgoritmoGeneticotoNXT();
```

Figura 20 – Declaração de variáveis e objetos universais da classe PCtoNXT.java (Autor)

O método principal da classe “PCtoNXT.java” é responsável por iniciar o programa e apresentar duas opções ao usuário, como demonstrado na figura 21. O método principal da classe “PCtoNXT.java” apresenta um menu de opções para o usuário, permitindo-lhe escolher o tipo de procedimento a ser iniciar: mapeamento de um ambiente com opção de sua locomoção ou apresentação gráfica.

```
public static void main(String[] args) throws IOException {
    System.out.println("Entrando no método \"PCtoNXT.main\"");
    Scanner input = new Scanner(System.in);

    System.out.printf("%s%s%s%s",
            "Escolha uma opção:\n",
            "1 - Mapear novo ambiente e salvar arquivo;\n",
            "2 - Carregar projeto e iniciar algoritmo de objetivos;\n\n",
            "Entre com sua opção: ");
    switch(input.nextInt()){
    case 1: opcao1();
            break;
    case 2: opcao2();
            break;
        default: System.out.printf("\n%s%s",
                "Opção não válida.",
                "Programa será fechado.");
                System.exit(0);
                break;
    }
}
```

Figura 21 – Método principal da classe PCtoNXT.java (Autor)

A opção 1, detalhada na figura 22, inicia um novo projeto e um novo mapeamento por conta do robô, assim como processamento por parte do Algoritmo Genético, criação de arquivos de registros e demais atividades. Já a opção 2 carrega um projeto já existente para apresentação da renderização (geração gráfica) de um projeto existente.

```
static private void opcao1() throws IOException{
    System.out.println("Entrando no método \"PCtoNXT.opcao1\"");
    Scanner input = new Scanner(System.in);
    dispositivoBluetooth.conectaBluetoothNXT();

    System.out.printf("Digite o nome do projeto: ");
    setNomeProjeto(input.nextLine());

    while(true){
        requestLeituraDistancias();
        algoritmoGenetico.iniciaAlgoritmoGenetico();
        algoritmoGenetico.proximoPasso();
    }
}
```

Figura 22 – Opção 1 da classe PCtoNXT.java (Autor)

Dentre os outros métodos da classe “PCtoNXT.java”, os mais importantes são:

• requestLeituraDistancias(): Envia ao robô comando para leitura das distancias ao seu redor e envia para trabalho pelo Algoritmo Genético;

• requestCaminhoXY(): Envia ao robô nova coordenada a ser avaliada.

Ambos implementados na figura 23.

```
static public void requestLeituraDistancias(){
    System.out.println("Entrando no método \"PCtoNXT.requestLeituraDistancias\"");
    dispositivoBluetooth.enviaMensagemBluetooth(5);
    {
        int j;
        for (j=0;j<numeroAngulosLidosRobo;j++){
            dispositivoBluetooth.enviaMensagemBluetooth(j);
            algoritmoGenetico.setDistancias(dispositivoBluetooth.recebeMensagemBluetooth(), j);
        }
        dispositivoBluetooth.enviaMensagemBluetooth(numeroAngulosLidosRobo+1);
    }
}

public void requestCaminhoXY(){
    System.out.println("Entrando no método \"PCtoNXT.requestCaminhoXY\"");
    dispositivoBluetooth.enviaMensagemBluetooth(3);
    dispositivoBluetooth.enviaMensagemBluetooth(proximoX);
    dispositivoBluetooth.enviaMensagemBluetooth(proximoY);
}
```

Figura 23 – Métodos “requestLeituraDistancias” e “requestCaminhoXY” (Autor)

4.2.2.2 DispositivoBluetooth.java

A classe “DispositivoBluetooth.java” é responsável pela criação e destruição de um canal de comunicação Bluetooth com o micro-controlador NXT e, a partir deste, enviar e receber dados do robô.

Utilizando 3 classes da API Java e 1 classe da API LeJOS auto-explicativas apresentadas na figura 24, a classe DispositivoBluetooth.java é capaz de encontrar um micro-controlador NXT à espera de uma conexão Bluetooth e criar um canal de comunicação sincronizado com o mesmo.

A classe DispositivoBluetooth.java contem 4 métodos auto-explicativos:

• conectaBluetoothNXT: Procura qualquer micro-controlador NXT com a função Bluetooth por perto e se conecta com o primeiro avistado;

• desconectarBluetoothNXT: Fecha todos os canais de comunicação e conexões ativas;

• enviaMensagemBluetooth: Envia um dado primitivo Java do tipo inteiro para o micro-controlador NXT conectado;

• recebeMensagemBluetooth: Recebe um dado primitivo Java do tipo inteiro do micro-controlador NXT conectado.

```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import lejos.pc.comm.NXTConnector;
```

Figura 24 – Bibliotecas importadas pela classe DispositivoBluetooth.java (Autor)

4.2.2.3 LogPCtoNXT.java

A classe LogPCtoNXT.java é responsável por criar, salvar e carregar arquivos de registros, assim como armazenar informações da iteração processada pelo Algoritmo Genético, como informações básicas do robô (coordenadas cartesianas e ângulo de face) e dados de trabalho (resultados de iterações executadas pelo Algoritmo Genético que precisam serem lidas constantemente). Para tal, a classe LogPCtoNXT.java utiliza 4 classes importadas da API Java e declara uma variedade enorme de variáveis de trabalho, como é demonstrada na figura 25.

```
import java.io.FileNotFoundException;
import java.io.File;
import java.io.IOException;
import java.io.RandomAccessFile;

public class LogPCtoNXT {
    RandomAccessFile fileProjeto;//0
    RandomAccessFile fileReproducao;//1
    RandomAccessFile fileCrossover;//2
    String[] stringFiles = new String[3];
    final int diferencaAnguloMedicao = 10;
    final int numeroAngulosLidosRobo = 360/diferencaAnguloMedicao;
    private int numeroRegistro;
    private int posicaoRoboX;
    private int posicaoRoboY;
    private int anguloFaceRobo;
    private int iteracao;
    private int fitnessFinal;
    private int anguloFinal;
    private int distanciaFinal;
    private int[] coordenadasFinais = new int[2];//X2 e Y2 finais
    private int [] distancias = new int [numeroAngulosLidosRobo];
    private int [] fitness = new int [numeroAngulosLidosRobo];
    private int [] porcentagemIncidenciaIndividuo = new int [numeroAngulosLidosRobo];
    private int [] resultadoAngulosIteracao = new int [numeroAngulosLidosRobo];
    private int [] resultadoFitnessIteracao = new int [numeroAngulosLidosRobo];
    private int [] resultadoFitnessCrossover = new int [numeroAngulosLidosRobo];
    private int [][] vetoresMedidos = new int [numeroAngulosLidosRobo][4];
```

Figura 25 – Bibliotecas importadas e variáveis e instancias universais criadas pela classe LogPCtoNXT.java (Autor)

Todos os métodos da classe LogPCtoNXT.java são auto-explicativos e, em sua maioria, são somente de entrada e recuperação de dados. A classe gera três arquivos de registros independentes no formato de acesso randômico, sendo o mais importante deles o arquivo "projeto.txt".

Cada arquivo de registro possui uma arquitetura diferente e um objetivo diferente. O arquivo de registro do projeto é o mais importante dos três, é nele que ficam armazenadas todas as informações sobre as distancias lidas, posições cartesianas dos pontos avaliados e resultados finais de cada iteração. Sua estrutura básica, gerada a cada iteração, é apresentada no quadro 3, a seguir:

Quadro 3 – Formato de dados armazenados por iteração nos registros do projeto

<table>
<tr><td>Número do registro (Iteração do Algoritmo Genético) – 8 bytes</td><td>Posição X atual da iteração – 8 bytes</td><td>Posição y atual da iteração – 8 bytes</td><td>Angulo de face do robô da iteração – 8 bytes</td><td>Diferença de ângulo entre uma medição e outra – 4 bytes</td></tr>
<tr><td colspan="5">Vetor contendo informações das distancias medidas e seus respectivos valores de aptidão – 576 bytes</td></tr>
<tr><td colspan="2">Angulo da próxima coordenada a ser medida até a coordenada atual – 8 bytes</td><td colspan="2">Distancia entre a coordenada a ser medida e a coordenada atual – 8 bytes</td><td>Valor de aptidão escolhido na iteração atual – 8 bytes</td></tr>
<tr><td colspan="5">Vetor contendo todos os vetores medidos (coordenadas de origem e coordenadas finais) – 1296 bytes</td></tr>
<tr><td colspan="2">Posição X do próximo ponto a ser avaliado – 8 bytes</td><td colspan="3">Posição Y do próximo ponto a ser avaliado – 8 bytes</td></tr>
<tr><td colspan="5">Total de bytes gravados por iteração: 1948 bytes</td></tr>
</table>

(Autor)

A estrutura de dados dos arquivos de registros dos procedimentos de reprodução e *crossover* se assemelham, pois ambos são responsáveis por armazenar os resultados finais dos respectivos processos durante cada iteração e essa informação pode ser usada mais tarde para uma avaliação de desempenho do algoritmo em certos ambientes (fugindo do foco deste projeto de pesquisa). A estrutura básica de cada iteração do arquivo de registros dos resultados do procedimento de reprodução encontra-se no quadro 4.

Quadro 4 – Formato de dados armazenados por iteração nos registros da reprodução

Número do registro (Iteração do Algoritmo Genético) – 8 bytes
Vetor contendo a porcentagem de incidência de cada ângulo avaliado – 288 bytes
Número de iterações executadas pelo processo de reprodução – 8 bytes
Vetor contendo os Ângulos e Valores de aptidão provenientes da reprodução – 576 bytes
Total de bytes gravados por iteração: 680 bytes

(Autor)

A estrutura básica de cada iteração do arquivo de registros dos resultados do procedimento de *crossover* encontra-se no quadro 5.

Quadro 5 – Formato de dados armazenados por iteração nos registros do crossover

Número de registro (Iteração do Algoritmo Genético) – 8 bytes
Vetor contendo os novos valores de aptidão provenientes da reprodução – 288 bytes
Total de bytes gravados por iteração: 296 bytes

(Autor)

Como é possível notar, não existe um registro para armazenamento de dados provenientes da mutação. Isto é devido a inexistência do procedimento de mutação no Algoritmo Genético Híbrido utilizado neste projeto. A função principal do procedimento de mutação é impedir que o Algoritmo Genético aponte rapidamente para uma solução, porém esse cuidado é efetuado a partir de um aumento de iterações nos procedimentos de reprodução e *crossover* do Algoritmo Genético. Além do mais, o processo de mutação poderia impedir a avaliação de um ângulo destino final ideal.

4.2.2.4 AlgoritmoGeneticotoNXT.java

A classe AlgoritmoGeneticotoNXT.java é a mais importante deste projeto de pesquisa. Ela é responsável por processar os dados recebidos pelo robô e enviar-lhe comandos, assim como preencher os arquivos de registros que são utilizados posteriormente para a procura de novos pontos desconhecidos e pelo algoritmo de processamento gráfico.

É abordada a explicação somente dos métodos principais desta classe, tendo em vista que a mesma contém muitas outros metodo de tratamento matemático algébrico e geométrico que se encontram facilmente em (STEINBRUCH, 1987) e (BOLDRINI, 1980)

Como é apresentado na figura 26, o trajeto resumido do algoritmo genético é:

1. Iniciar o projeto criando os arquivos de registros na pasta do projeto;
2. Preparar a classe "LogPCtoNXT.java" para recebimento de dados;
3. Processamento do cálculo de aptidão do espaço de amostragem;
4. Procedimento de reprodução e *crossover*;
5. Checagem dos valores obtidos;
6. Obtenção da próxima coordenada a ser avaliada;
7. Fechamento dos arquivos de registros.

Com essa estrutura básica o Algoritmo Genético é capaz de processar os dados recebidos e requisitar a leitura de mais dados em uma coordenada distinta ou finalizar o programa.

```
public void iniciaAlgoritmoGenetico() throws IOException{
    System.out.println("Entrando no método \"AlgoritmoGeneticoto.iniciaAlgoritmoGenetico\"");
    int [] fitnessFinal = new int [this.numeroAngulosLidosRobo];
    int i;

    if(logNXT.getNumeroRegistro()==0){
        logNXT.iniciaProjeto(controlador.getNomeProjeto());
    }
    logNXT.resetaDadosLog();
    this.atualizaRegistroLog();
    this.transformaDistanciasAtuaisFitnessNow();

    do{
        this.reproducaoAlgoritmoGenetico();
        this.crossoverAlgoritmoGenetico();

        for (i=0;i<this.numeroAngulosLidosRobo;i++){
            fitnessFinal[i]=this.fitnessNowAngulosMedidos[0][i];
        }

        this.getSomaFitness(fitnessFinal);
        this.getSomaAngulos(fitnessNowAngulosMedidos[1]);
    }while(this.getSomaFitness()/this.numeroAngulosLidosRobo!=fitnessFinal[0] &&
            this.getSomaAngulos()/this.numeroAngulosLidosRobo!=fitnessNowAngulosMedidos[1][0]);

    logNXT.setFitnessFinal(fitnessFinal[0]);
    logNXT.setAnguloFinal(fitnessNowAngulosMedidos[1][0]);
    avaliaDistanciaFinal();
    logNXT.salvaLogProjeto();
}
```

Figura 26 – Método de inicio do Algoritmo Genético da classe AlgoritmoGeneticotoNXT.java (Autor)

Um fator determinante para o bom funcionamento de um Algoritmo Genético é a formalização de sua função objetivo, ou seja, a função matemática que irá converter os dados do espaço de amostragem em seus valores de aptidão para serem processados. Esse trabalho é feito pelo método "transformaDistanciasAtuaisFitnessNow" que, além de aplicar a função objetivo a todas as distancias medidas, ainda avalia nos registros do projeto as coordenadas já conhecidas para diminuir o número de pontos no espaço físico avaliados mais de uma vez. O código-fonte da já mencionada função objetivo do Algoritmo Genético deste projeto se encontra na figura 27.

O método "transformaDistanciasAtuaisFitnessNow" utiliza de artifício uma das características da API LeJOS contida no algoritmo de controle de robô: a medição de distância pelo sensor ultra-sônico utilizando a API LeJOS tem um limite seguro de 170 centímetros, porém esta API retorna o valor 255 caso não tenha sido possível medir a distância entre o sensor e o objeto (infelizmente este valor também pode ser retornado como um erro na leitura de superfícies não uniformes ou em objetos com distâncias muito próximas, como de 5 centímetros para baixo). Utilizando desse artifício para a definição dos valores de

aptidão do espaço de amostragem, a função objetivo descarta as medições efetuadas com sucesso trabalhando somente com as medições que retornaram o valor 255, ou seja, que tem grande chances de ser uma área inexplorada pelo robô. Além disso, para assegurar que naquele ângulo exista uma possível área ainda não conhecida, é utilizado o conhecimento dos ângulos vizinhos para julgar se ali se encontram realmente pontos no espaço físico ainda não avaliados.

```
public void transformaDistanciasAtuaisFitnessNow() throws IOException{
    System.out.println("Entrando no método \"AlgoritmoGeneticoto.transformaDistanciasAtuaisFitnessNow\"");
    int i,j;
    int [] fitnessTemp = new int [this.numeroAngulosLidosRobo];
    this.resetaFitnessNow();
    for (i=0;i<this.numeroAngulosLidosRobo;this.setFitnessNow((this.getDistancias(i)), i),i++);

    j=0;
    do{
        for (i=0;i<this.numeroAngulosLidosRobo;i++){
            fitnessTemp[i]=this.getFitnessNow(i-1<0?this.numeroAngulosLidosRobo-1:i-1)+
                    this.getFitnessNow(i)+
                    this.getFitnessNow(i+1>=this.numeroAngulosLidosRobo?0:i+1);
            if(this.getFitnessNow(i)<2500)
                fitnessTemp[i]=0;
        }
        for (i=0;i<this.numeroAngulosLidosRobo;this.setFitnessNow(fitnessTemp[i],i),i++);
        j++;
    }while(j!=2);

    if(logNXT.getNumeroRegistro()!=1)
        this.comparaFitnessNowFitnessAll();

    //Atualiza log com Fitness Iniciais
    for (i=0;i<this.numeroAngulosLidosRobo;logNXT.setFitness(this.getFitnessNow(i), i),i++);
}
```

Figura 27 – Código-fonte da função objetivo do Algoritmo Genético, presente na classe AlgoritmoGeneticotoNXT.java (Autor)

Ao final do processamento da função objetiva, os valores obtidos são passados para análise dos pontos já medidos contidos nos registros. Caso a função objetiva contenha algum ponto já analisado, seu valor de aptidão é anulado fazendo com que tenha uma taxa de incidência muito baixa na roleta de reprodução.

Após os procedimentos de reprodução e *crossover* do Algoritmo Genético, é obtido um ângulo definitivo do próximo ponto a ser avaliado no espaço físico. As coordenadas em plano cartesiano presentes na direção desse ângulo são calculadas pelo método "avaliaDistanciaFinal", apresentado nas figuras 28, 29 e 30, que avalia a distância máxima segura a ser percorrida naquela direção de acordo com dados armazenados nos registros.

```
public void avaliaDistanciaFinal() throws IOException{
    System.out.println("Entrando no método \"AlgoritmoGeneticoto.avaliaDistanciaFinal\"");
    int x,distanciaTemp,j,z,distanciaMaxima;
    int coordenadaX,coordenadaY,proximoX,proximoY;;
    int [][] vetoresCadastrados = new int [this.numeroAngulosLidosRobo][4];

    x=logNXT.getAnguloFinal();

    do{
        if(x<0)
            x=x+360;
        else if(x>=360)
            x=360-x;
    }while(!(x>=0 && x<360));

    distanciaMaxima=this.getDistancias((x-logNXT.getAnguloFaceRobo())/this.diferencaAnguloMedicao);

    if(logNXT.getNumeroRegistro()>1){
        for (distanciaTemp=0;distanciaTemp<1700;distanciaTemp+=10){

            if(x>=0 && x<90){
                coordenadaX=(int)(distanciaTemp*Math.cos(Math.toRadians(x)))+logNXT.getPosicaoRoboX();
                coordenadaY=(int)(distanciaTemp*Math.sin(Math.toRadians(x)))+logNXT.getPosicaoRoboY();
            }else if (x>=90 && x<180){
                coordenadaX=-(int)(distanciaTemp*Math.cos(Math.toRadians(180-x)))+logNXT.getPosicaoRoboX();
                coordenadaY=(int)(distanciaTemp*Math.sin(Math.toRadians(180-x)))+logNXT.getPosicaoRoboY();
            }else if(x>=180 && x<270){
                coordenadaX=(int)(distanciaTemp*Math.cos(Math.toRadians(x)))+logNXT.getPosicaoRoboX();
                coordenadaY=(int)(distanciaTemp*Math.sin(Math.toRadians(x)))+logNXT.getPosicaoRoboY();
            }else{
                coordenadaX=(int)(distanciaTemp*Math.cos(Math.toRadians(-(360-x))))+logNXT.getPosicaoRoboX();
                coordenadaY=-(int)(distanciaTemp*Math.sin(Math.toRadians(-(360-x))))+logNXT.getPosicaoRoboY();
            }
```

Figura 28 – Método “avaliaDistanciaFinal” da classe AlgoritmoGeneticotoNXT.java - Parte 1 (Autor)

Caso naquela direção não exista nenhuma área desconhecida e o Algoritmo Genético aponte um possível erro, um método para descobrimento de pontos desconhecidos é chamado.

O método “procuraPontoDesconhecido2” é responsável por análise de todos os registros do projeto e movimentação do robô até o novo ponto a ser analisado. Caso o mesmo não encontre nenhum ponto aparentemente desconhecido, envia comando para o robô se auto-desligar, fecha as conexões Bluetooth, salva os registros do projeto e encerra o aplicativo.

```
        for(j=1;j<logNXT.getNumeroRegistro();j++){
            vetoresCadastrados=logNXT.carregaPontosLidosProjeto(j);
            for(z=0;z<this.numeroAngulosLidosRobo;z++){
                int X1,X2,X3,X4,Y1,Y2,Y3,Y4;
                X1=vetoresCadastrados[z][0];
                X2=vetoresCadastrados[z][1];
                X3=vetoresCadastrados[(z+1)>=this.numeroAngulosLidosRobo?z-this.numeroAngulosLidosRobo+1:z+1][1];
                X4=coordenadaX;
                Y1=vetoresCadastrados[z][2];
                Y2=vetoresCadastrados[z][3];
                Y3=vetoresCadastrados[(z+1)>=this.numeroAngulosLidosRobo?z-this.numeroAngulosLidosRobo+1:z+1][3];
                Y4=coordenadaY;

                if(!((X1==X2 && Y1==Y2) || (X1==X3 && Y1==Y3)))
                    if(this.checaSePontoEstaDentroDoTriagulo(X1,Y1,X2,Y2,X3,Y3,X4,Y4)){
                        if(distanciaMaxima>distanciaTemp)
                            distanciaMaxima=distanciaTemp;
                        break;
                    }
            }
        }
    }
}
```

Figura 29 – Método “avaliaDistanciaFinal” da classe AlgoritmoGeneticotoNXT.java – Parte 2

(Autor)

```
    distanciaMaxima=distanciaMaxima>1600?1600:distanciaMaxima;

    if(distanciaMaxima<10)
        procuraPontoDesconhecido2();
    else{
        logNXT.setDistanciaFinal(distanciaMaxima-200>=0?distanciaMaxima-200:distanciaMaxima);

        distanciaMaxima=logNXT.getDistanciaFinal();


        if(x>=0 && x<90){
            proximoX=(int)((double)distanciaMaxima*Math.cos(Math.toRadians(x)))+logNXT.getPosicaoRoboX();
            proximoY=(int)((double)distanciaMaxima*Math.sin(Math.toRadians(x)))+logNXT.getPosicaoRoboY();
        }else if (x>=90 && x<180){
            proximoX=-(int)((double)distanciaMaxima*Math.cos(Math.toRadians(180-x)))+logNXT.getPosicaoRoboX();
            proximoY=(int)((double)distanciaMaxima*Math.sin(Math.toRadians(180-x)))+logNXT.getPosicaoRoboY();
        }else if(x>=180 && x<270){
            proximoX=(int)((double)distanciaMaxima*Math.cos(Math.toRadians(x)))+logNXT.getPosicaoRoboX();
            proximoY=(int)((double)distanciaMaxima*Math.sin(Math.toRadians(x)))+logNXT.getPosicaoRoboY();
        }else{
            proximoX=(int)((double)distanciaMaxima*Math.cos(Math.toRadians(-(360-x))))+logNXT.getPosicaoRoboX();
            proximoY=-(int)((double)distanciaMaxima*Math.sin(Math.toRadians(-(360-x))))+logNXT.getPosicaoRoboY();
        }

        logNXT.setCoordenadasFinaisX(proximoX);
        logNXT.setCoordenadasFinaisY(proximoY);
    }
}
```

Figura 30 – Método “avaliaDistanciaFinal” da classe AlgoritmoGeneticotoNXT.java – Parte 3

(Autor)

4.2.3 Algoritmo de processamento gráfico

O algoritmo de processamento gráfico é composto de somente uma classe que interage com as classes “LogPCtoNXT.java” e “PCtoNXT.java” do algoritmo de processamento

lógico. Essa classe é a “TesteSwing.java” e utiliza como artifício de geração gráfica a tecnologia *Java Swing*.

A classe “TesteSwing.java” não possui nenhum tratamento especial para a apresentação gráfica do mapeamento do objeto, tornando desnecessária sua demonstração aprofundada.

CAPÍTULO 5 – TESTES E RESULTADOS

5.1 Primeiro protótipo da arquitetura do robô

A primeira arquitetura idealizada para o robô consistia na utilização de três servomotores trabalhando independentemente, cada um com sua própria função, como mostra a figura 31.

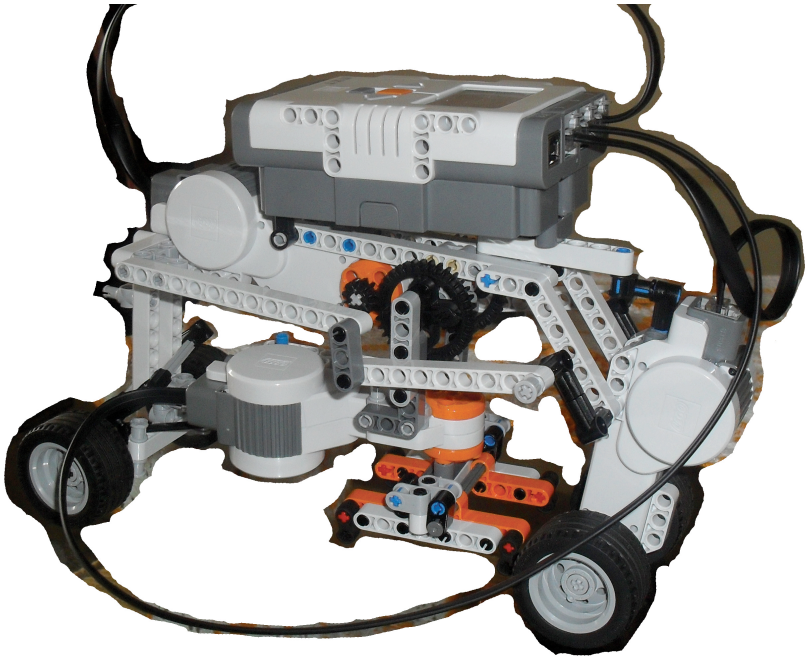

Figura 31 – Primeiro protótipo do robô (Autor)

Um dos servomotores era responsável pela movimentação retilínea do robô, tanto para frente quanto para trás; um segundo servomotor, responsável pela levitação do robô para que, por último, um outro servomotor pudesse rotacioná-lo em volta do seu próprio eixo.

O primeiro protótipo do robô apresentava muitas dificuldades em sua estrutura, que acabou sendo abandonada sem ser finalizada.

Na figura 32, o protótipo não apresentava o sensor ultra-sônico e o micro-controlador NXT instalados.

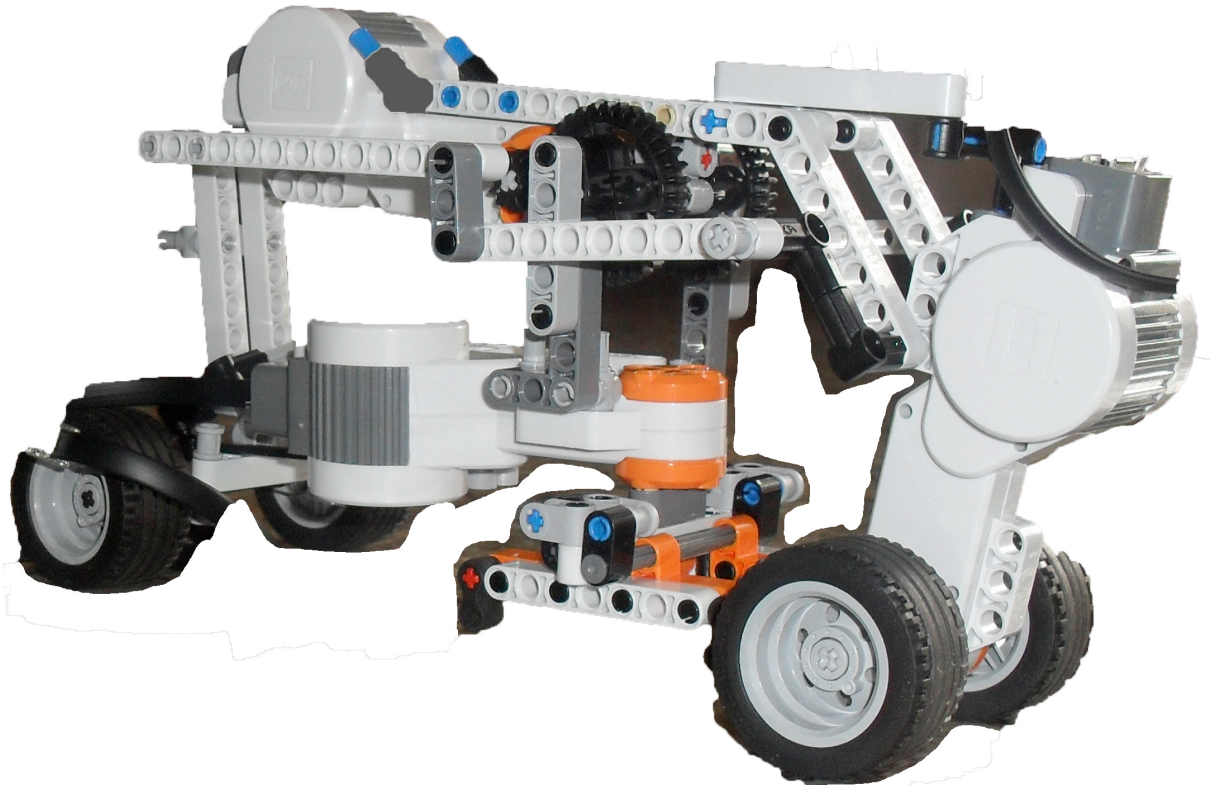

Figura 32 – Primeiro protótipo do robô. Sem instalação do sensor ultra-sônico e do micro-controlador NXT (Autor)

5.2 Teste de leitura de distância em uma área fechada

Em vista aos problemas de medição apresentados pelo sensor ultra-sônico, esse teste busca demonstrar as suas limitações na aplicação sugerida por este projeto de pesquisa.

Nesse teste, o robô é posicionado em uma área fechada com somente um opção de saída. O Algoritmo Genético faz a requisição da leitura de distâncias e o resultado é capturado pelo debug da aplicação, apresentado na figura 33. A partir desses resultados é possível ver que, mesmo com as limitações, o robô é capaz de reconhecer grande parte das distâncias entre ele e os objetos a sua volta. Também é possível perceber que certos objetos mais próximos o atrapalham na medição de objetos mais distantes, fazendo-o descartar a medição ou usar valores de distância do objeto mais próximo.

[0]	0
[1]	2550
[2]	2550
[3]	2550
[4]	2550
[5]	2550
[6]	2550
[7]	380
[8]	370
[9]	370
[10]	370
[11]	370
[12]	390
[13]	2550
[14]	2550
[15]	2550
[16]	2550
[17]	2550
[18]	2550
[19]	2550
[20]	2550
[21]	2550
[22]	540
[23]	520
[24]	520
[25]	530
[26]	2550
[27]	2550
[28]	2550

Figura 33 – Debug da aplicação retornando valores lidos pelo robô (Autor)

5.3 Teste de leitura de ambiente

O teste a seguir foi feito a partir de um canto de uma sala avaliando a leitura de dados corretamente por parte do robô e processamento de dados corretamente por parte do algoritmo de processamento lógico.

Ao iniciar o aplicativo, o Algoritmo Genético faz a requisição das distâncias ao redor do robô. A partir do debug da aplicação é possível verificar esses dados, apresentados na figura 35. Os dados apresentados na figura 34 (“x” é o ângulo a que o robô irá locomover-se; “distanciaMaxima” é o valor da distância percorrida pelo robô em milímetros, e “proximoX” e “proximoY” formam as coordenadas cartesianas do próximo ponto a ser avaliado) são resultados dessa primeira iteração .

x	70
distanciaMaxima	1400
proximoX	478
proximoY	1315

Figura 34 – Resultados finais da iteração do processamento do Algoritmo Genético (Autor)

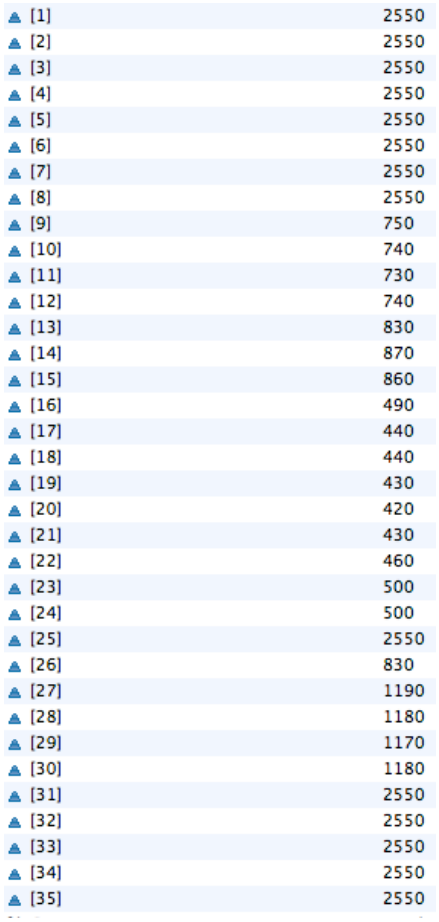

Figura 35 – Debug da aplicação apresentando distâncias medidas em milímetros (Autor)

Nesse momento o Algoritmo Genético faz a requisição de uma nova leitura de distâncias para ser processada novamente, desta vez checando os registros para a tentativa de evitar espaços já conhecidos, figura 37. Esse procedimento é repetido até que o método de busca de ponto desconhecido não encontre mais esses pontos. Os resultados dessa última iteração podem ser visto na figura 34

Figura 34 – Resultados finais da iteração do processamento do Algoritmo Genético (Autor)

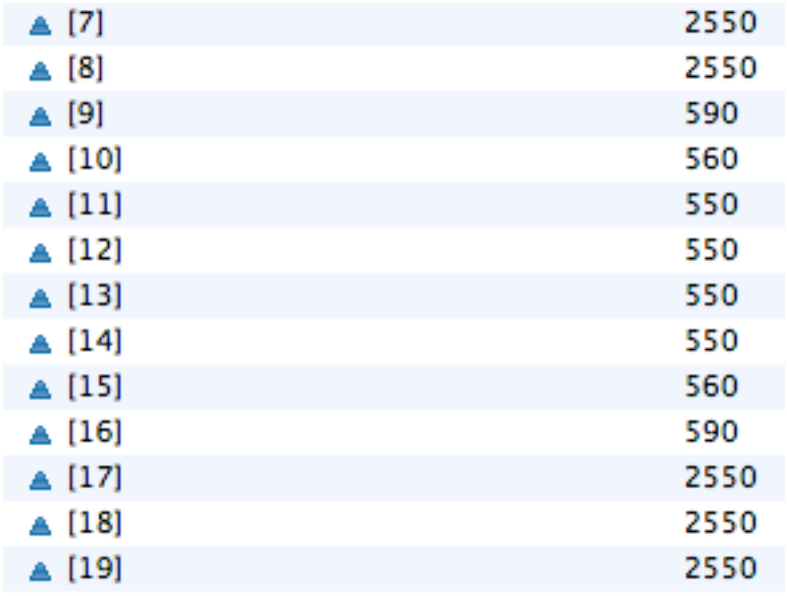

Figura 37 – Debug da aplicação apresentando distâncias medidas em milímetros (Autor)

Para efeito de exemplo de registros do projeto verificado pelo Algoritmo Genético, a figura 38 apresenta uma tela de debug com amostragens de vetores cadastrados da primeira iteração desse teste.

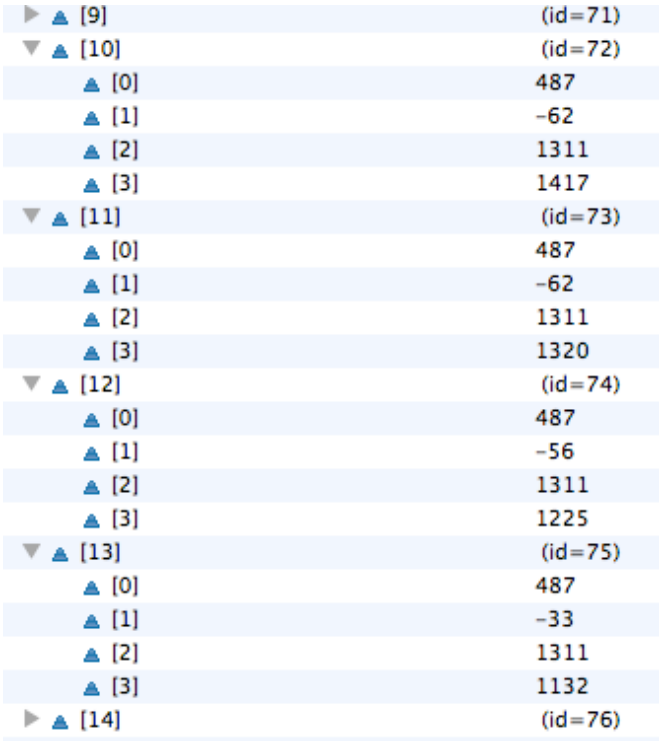

Figura 38 – Debug da aplicação apresentando vetores cadastrados no log. Posições X1,X2,Y1,Y2 dos vetores (Autor)

5.4 Teste de mapeamento de ambiente

Neste teste, o robô foi encarregado de mapear metade de um ambiente que continha um objeto em seio meio. O teste foi interrompido manualmente apos 40 minutos de processamento – provando que o mapeamento pode exigir um grande custo temporal devido ao recálculo de informações para garantir a segurança dos dados retirados do ambiente – e o algoritmo de renderização (geração gráfica) foi ativado para apresentar o resultado obtido até o momento.

Em um primeiro teste, apresentado na figura 39, percebe-se que o robô conseguiu trabalhar sempre com os dados corretos do ambiente e que após cerca de mais 20 minutos o mesmo poderia ter conseguido mapeado o ambiente por completo.

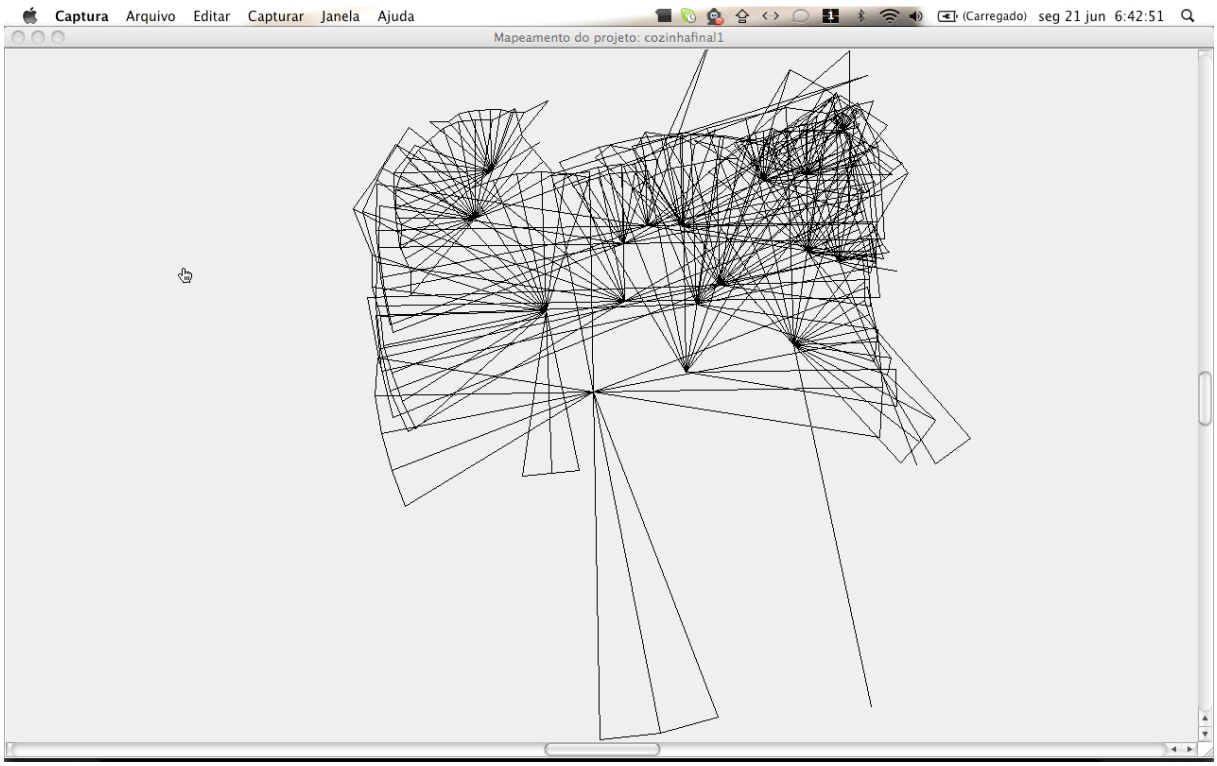

Figura 39 – Apresentação gráfica do mapeamento de um ambiente (Autor)

Na figura 39, é possível perceber que o robô foi capaz de mapear praticamente metade do ambiente, ainda identificando a localização do objeto no meio do mesmo e a distancia até

todas as paredes. Preenchendo a área conhecida para melhor visualização, consegue-se um idéia melhor da área mapeada pelo robô, apresentada na figura 40.

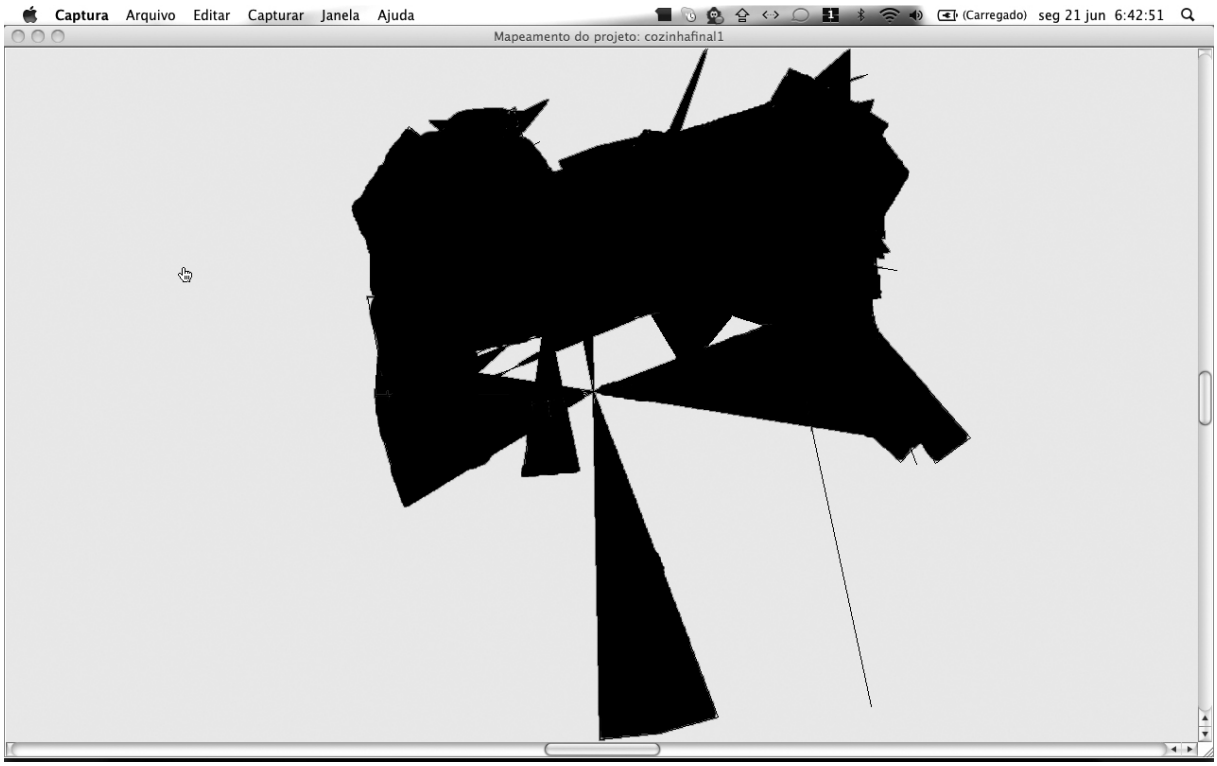

Figura 40 – Apresentação gráfica do mapeamento de um ambiente – Área conhecida preenchida de cor preta(Autor)

Efetuando um teste de mapeamento no mesmo ambiente com início no mesmo ponto de partida, a figura 41 apresenta claramente uma disparidade em comparação ao resultado obtido anteriormente, resultado dos erros decorrentes das limitações do sensor ultra-sônico o kit *Lego Mindstorm NXT*. Com uma análise mais aprofundada da figura 41, é possível perceber os erros que uma leitura errada de uma distância pode ocasionar ao algoritmo. O Algoritmo Genético não tem como saber se o robô lhe enviou dados certos ou não, efetuando seus procedimentos e apresentando um resultado assim mesmo. Geralmente, um mapeamento com erros é constatado a partir do momento que o robô se choca com algum obstáculo, provando que o mesmo contem informações de posicionamento e distancias diferentes dos reais.

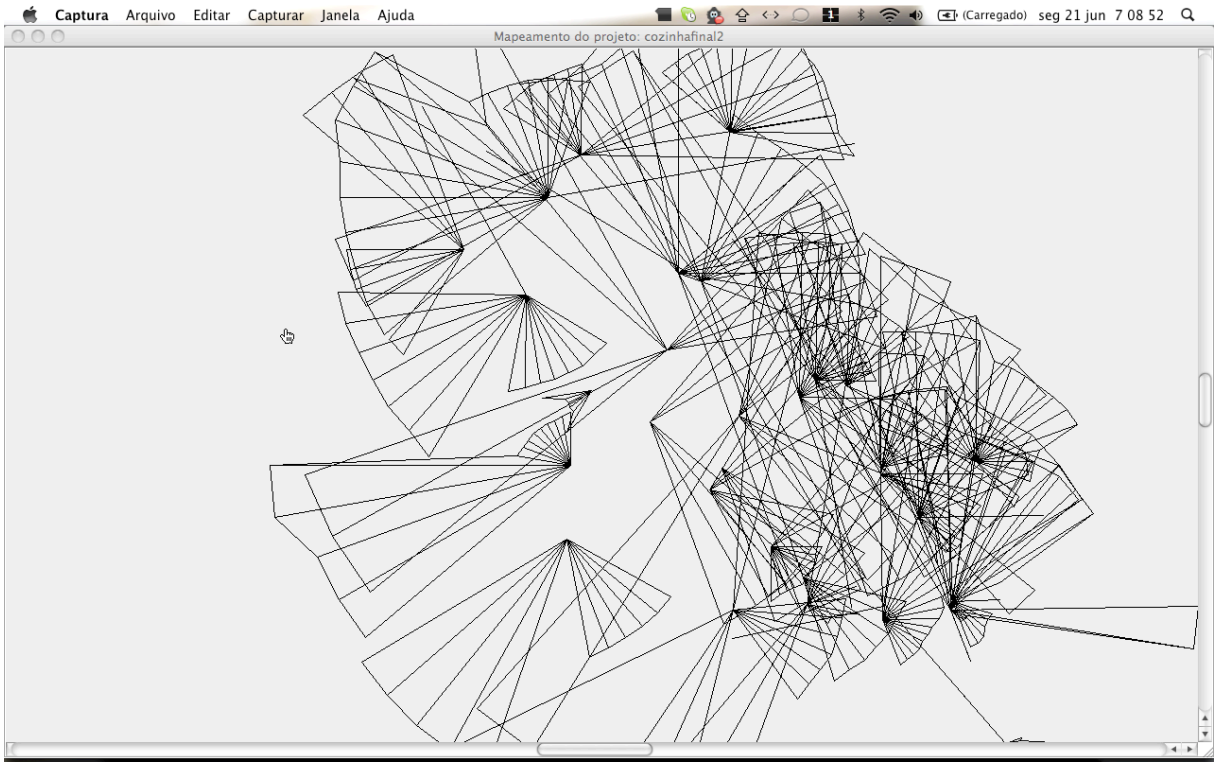

Figura 41 – Apresentação gráfica de um mapeamento de ambientes com erros. (Autor)

5.5 Dificuldades

Durante o desenvolvimento do projeto foram encontradas várias dificuldades que atrasaram a concepção do projeto e limitaram os resultados obtidos.

5.5.1 Bluetooth

Uma das principais dificuldades foi lidar com o Bluetooth e sua pilha de protocolos, tanto do chip contido no micro-controlador quanto o usado pelo computador.

Os testes com Bluetooth foram massivos e desestimulantes, atrasando o andamento do trabalho em dias.

5.5.2 Compilador NBC/NXC

O projeto foi planejado para, inicialmente, utilizar as linguagens de programação NBC/NXC no micro-controlador NXT. Porém o compilador Bricx (o qual é o encarregado para compilar os códigos em NBC/NXC), até a data do desenvolvimento deste projeto de pesquisa, ainda se encontrava em estado beta, tendo várias funções de controle do Bluetooth do micro-controlador NXT não reconhecidas.

5.5.3 Linguagem de programação a ser escolhida

O projeto começou a ser implementado utilizando as linguagens de programação NBC/NXC para o micro-controlador NXT e C/C++/Obj-C para o software de controle no computador. Porém, por diversos fatores, ambas não foram capazes de implementar as funcionalidades de conexão via Bluetooth entre o MacBook e o micro-controlador NXT. A partir daí, foram testadas várias outras linguagens de programação (RobotC, LabVIEW e NXT-G) porém nenhuma delas implementava a conexão Bluetooth e sua pilha de protolocos necessárias para a criação desse projeto, até a data da sua conclusao. Para solucionar o problema, foi instalado o firmware LeJOS versão 0.83 no micro-controlador (que é uma mini máquina virtual Java) e a biblioteca Bluecove no computador (que é uma biblioteca de gerenciamento da pilha de protocolos do Bluetooth para Java).

5.5.4 Limitações do sensor ultra-sônico do kit Lego Mindstorm NXT 2.0

De acordo com testes e pesquisas realizados com o sensor ultra-sônico do kit *Lego Mindstorm NXT* nos laboratórios de rede e engenharia da computação do Instituto Tecnológico de Zurique, Suíça (ETH, 2010), o sensor apresenta várias tendências a erros dependendo da distancia e ângulo do objeto medido.

De acordo com a figura 42, medidas avaliadas pelo sensor com uma distância real de 3 centímetros ou menos tem uma alta taxa de desvio, impossibilitando sua medição. Nos casos restantes, o desvio de medição permanece em um valor menor que 5 milímetros, garantindo a relevância dos dados obtidos. (ETH, 2010).

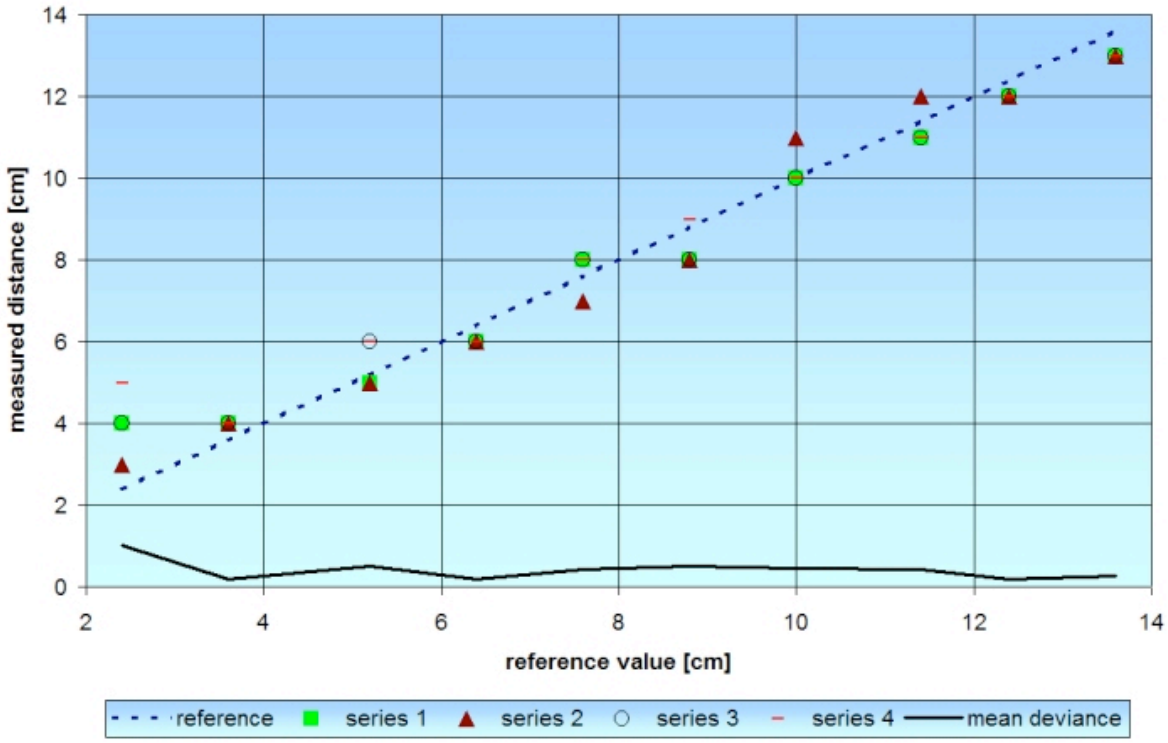

Figura 42 – Gráfico comparativo entre a distancia medida pelo sensor e seu real valor (ETH. Disponível em <http://www.tik.ee.ethz.ch/mindstorms/sa_nxt/index.php?page=tests_us>. Acessado 4 de Junho de 2010)

Os resultados obtidos nas figuras 43 e 44 mostram que o sensor ultra-sônico sempre deve estar na posição horizontal, pois outras posições reduzem bastante o campo de visão do sensor. Outro fato importante é que o sensor aparenta ser um pouco “cego” no seu olho esquerdo, o que pode ser explicado pelo fato de ser este olho o receptor do ultra-som, enquanto o olho direito é o emissor. (ETH, 2010).

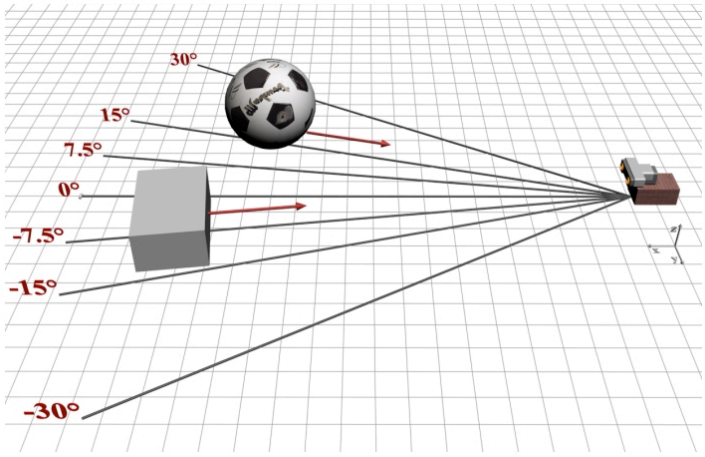

Figura 43 – Experimento de medição de vários ângulos com o sensor na posição horizontal (ETH. Disponível em <http://www.tik.ee.ethz.ch/mindstorms/sa_nxt/index.php?page=tests_us>. Acessado 4 de Junho de 2010)

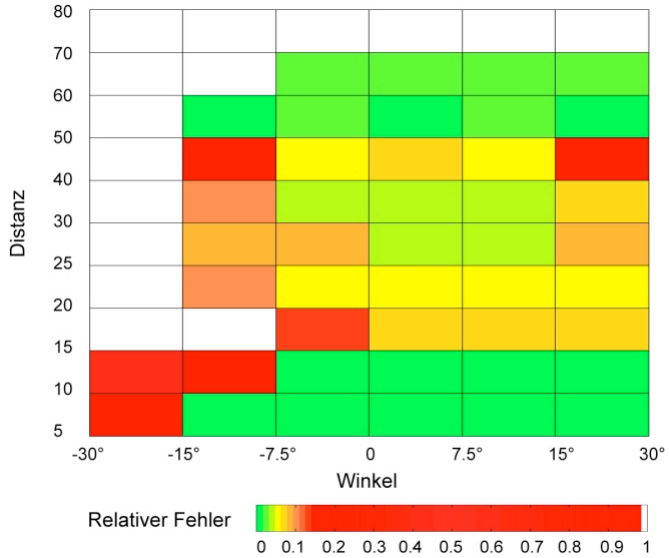

Figura 44 – Representação gráfica do campo de visão do sensor ultra-sônico na posição vertical (ETH. Disponível em <http://www.tik.ee.ethz.ch/mindstorms/sa_nxt/index.php?page=tests_us>. Acessado 4 de Junho de 2010)

O teste dinâmico com o sensor ultra-sônico demonstrado na figura 45 revela dois pontos fracos deste. O primeiro é que o sensor tende a medir 255 centímetros (valor retornado quando não encontrado obstáculo) no lugar da distância real em algumas áreas. O segundo, ainda mais crítico, encontra-se na área entre 25 e 50 centímetros, onde o sensor possui uma alta probabilidade de retornar o valor equivocado de 48 centímetros. (ETH, 2010).

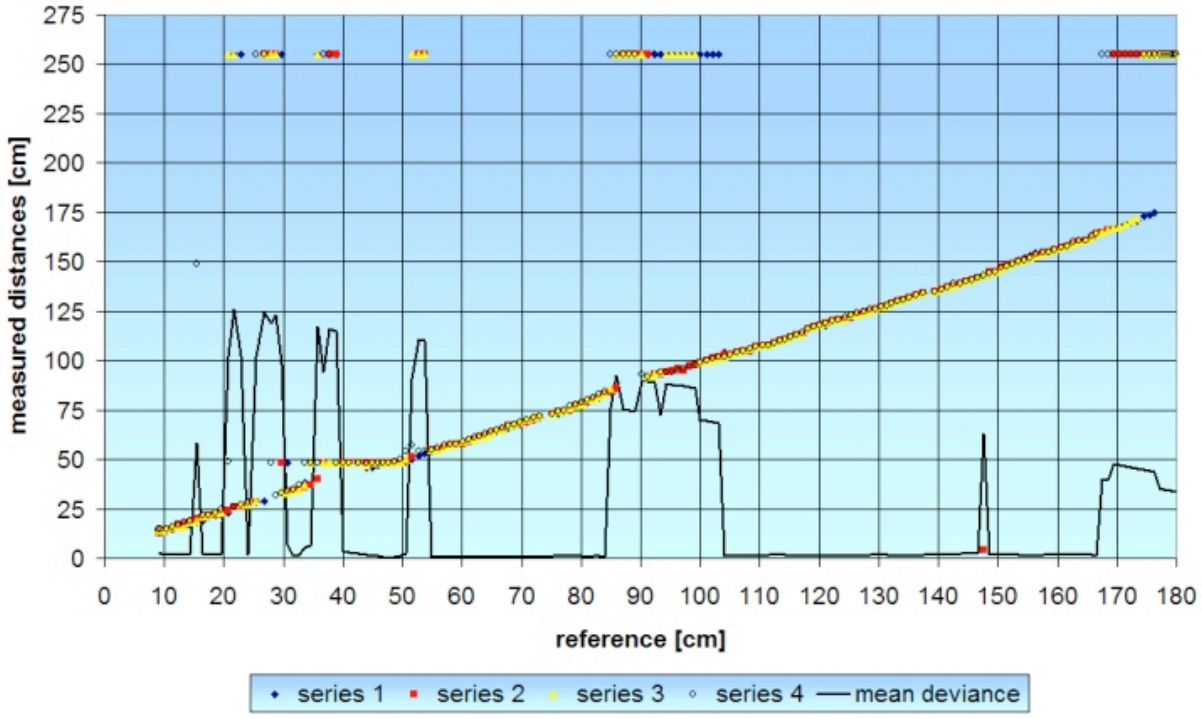

Figura 45 – Teste dinâmico com o sensor-ultrasônico enquano se aproxima de uma parede (ETH. Disponível em <http://www.tik.ee.ethz.ch/mindstorms/sa_nxt/index.php?page=tests_us>. Acessado 4 de Junho de 2010)

Esses problemas conseqüentemente irão ocasionar erros na decisão de ações do Algoritmo Genético deste projeto de pesquisa.

CAPÍTULO 6 - CONCLUSÃO

A utilização de Algoritmos Genéticos para o desenvolvimento de robôs móveis de baixo custo é totalmente viável, porém de alto nível de processamento para casos de robôs sem nenhum sensor de controle espacial – como um giroscópio e/ou acelerômetro – pois o computador central gasta muito poder matemático executando cálculos algébricos e geométricos para armazenamento e consultoria de dados.

Uma divisão de recursos seria o ideal e, com o uso de sensores instalados ao robô, o seu hardware final ainda pode ser de baixíssimo custo financeiro, como o apresentado neste projeto de pesquisa.

No caso da aplicação detalhada neste projeto de pesquisa, o uso de Algoritmos Genéticos para decisão de novas rotas se mostra muito eficiente e rápido, porém problemas encontrados no sensor medidor de distância ultra-sônico podem atrapalhar profundamente o sucesso dos cálculos do software. Esses problemas influenciam bastante no resultado final do mapeamento de um ambiente, pois necessita de várias leituras de dados a partir do sensor ultra-sônico, e uma simples leitura de dados errada pode fazer com que o Algoritmo Genético trabalhe com variáveis duvidosas, gerando, conseqüentemente, resultados duvidosos que podem alterar drasticamente o resultado final esperado.

Nas situações em que o sensor ultra-sônico emite dados corretos ao Algoritmo Genético, é fato este se mostra como uma ótima opção de determinador de ações, fornecendo resultados sempre corretos de uma maneira aleatória, ou seja, o resultado final de uma mesma análise de posição pode ser diferente para várias iterações, porém sempre apresentando um resultado que se encaixa no objetivo final.

O autor desse projeto conclui que a utilização de Algoritmos Genéticos na lógica de controle de robôs móveis se torna uma alternativa de fácil aplicabilidade e baixo investimento financeiro, considerando que o protótipo apresentado aqui somente utilizou o kit Lego Mindstorm NXT 2.0 (vendido ao preço de 279,99 dólares americanos pelo site oficial da Lego) (LEGO, 2010) e um computador pessoal. Porém, fica claro também que mais investimento em um número maior de sensores de alta qualidade apresentam um rendimento mais seguro e veloz à aplicação.

6.1 Sugestões de trabalhos futuros

Uma sugestão simples de trabalho futuro é a instalação de novos sensores a um robô automaticamente móvel terrestre como o apresentado neste projeto de pesquisa, como sensores medidores de distância de funcionamentos distintos, assegurando o valor dos dados retornados a partir de uma comparação, e a adição de sensores de movimento para diminuir o processamento de gestão posicional por parte do computador remoto.

Para estudiosos que buscam mais desafio, sugiro a utilização de Algoritmos Genéticos, ou outras técnicas genéticas, no controle em tempo real de robôs aerodinâmicos, como mikrokopter (MIKROKOPTER, 2010) onde o aluno passará a maior parte do seu tempo na implementação da Inteligência Artificial do robô sem precisar preocupar-se muito com seu hardware, com o auxílio de sensores de altitude, aceleração e posicionamento (GPS).

Trabalhos como esses são o ponta-pé inicial para pesquisas no ramo de rastreamento ou identificação de cenário.

REFERÊNCIAS BIBLIOGRÁFICAS

ASIMOV, Isaac. Eu, robô. Rio de Janeiro: Exped, 1969.

APPLE Developer, Mac OS X Reference Library. Apresenta guia prático para desenvolvimento de software utilizando o Cocoa framework. Disponível em <http://developer.apple.com/mac/library/documentation/Cocoa/Conceptual/CocoaFundamentals/WhatIsCocoa/WhatIsCocoa.html#//apple_ref/doc/uid/TP40002974-CH3-SW16> Acesso em 6 de Abril de 2010

APPLE Developer, Mac OS X Reference Library. Apresenta guia prático para geração de gráficos e animações no Mac OS X. Disponível em <http://developer.apple.com/mac/library/referencelibrary/GettingStarted/GS_GraphicsImaging/index.html> Acesso em 6 de Abril de 2010.

BAECK, Thomas, "Evolutionary Algorithms: Comparison of Approaches", in R. Paton (ed.) Computing with Biological Metaphors, Chapman & Hall, Capítulo 14, pp. 227-243, 1994.

BARR e FEIGENBAM. The Handbook of Artificial Intelligence, volume 1. 1 ed. William Kaufmann, Inc. 1981

BELLMAN, Richard E. An Introduciton to Artificial Intelligence: Can Computers Think?, 1 ed. Boyd & Frases Publishing Company. 1978

BLUETOOTH SIG. Página oficial do grupo de interesse especial em Bluetooth, o qual desenvolveu e licenciou a tecnologia. Apresenta informações técnicas e artigos sobre a tecnologia e seu uso. Disponível em <http://www.bluetooth.com/English/Pages/default.aspx>. Acesso em 15 Abril 2010.

BOLDRINI, José Luís. Álgebra Linear. 3 ed. São Paulo : Harper & Row do Brasil. 1980

BRICXCC. Página oficial das linguagems NBC/NXC. Apresenta exemplos, guias e bibliotecas para programação nas linguagens NBC/NXC. Disponível em <http://bricxcc.sourceforge.net/nbc/> Acesso em 03 de Maio de 2010.

CHARNIAK e MCDERMOTT. Introduction to Artificial Intellilgence. 1 ed. Addison-Wesley, Reading, MA. 1985

ETH, computer engineering and networks laboratory. Página dos laboratórios de redes e engenharia da computação do Instituto Federal de Tecnologia de Zurique. Apresenta dados analíticos sobre o sensor ultra-sônico do kit Lego Mindstorm NXT. Disponível em <http://www.tik.ee.ethz.ch/mindstorms/sa_nxt/index.php?page=tests_us> Acessado em 4 de Junho de 2010.

EVERYMAC. Site especializados em computadores pessoais da Apple. Disponível em <http://www.everymac.com/systems/apple/macbook_pro/stats/macbook-pro-core-2-duo-2.26-aluminum-13-mid-2009-sd-firewire-800-unibody-specs.html> Acesso em 13 de maio de 2010

FOGEL. Artificial Intelligence Throught Simulated Evolution. 1 ed. John Wiley & Sons, Inc, New York, 1996.

FOLHAONLINE, página oficial do jornal Folha. Disponível em http://www1.folha.uol.com.br/folha/informatica/ult124u638591.shtml. Acesso em 6 de Junho de 2010.

GOLDBERG, David E. Genetic Algorithms Search, Optimization, Machine Learning. 1 ed. Estados Unidos da América, 1953.

HOLLAND, John Henry. Adaptation in Natural and Artificial System. 2 ed. MIT Press. 1992

KOZA, John. Genetic Programming: On the Programming of Computer by Means of Natural Selection. 1 ed. MIT Press. 1992.

KURZWEIL, Ray. The Age of Intelligent Machines. 1 ed. MIT Press 1990

LEGO MINDSTORM NXT SOFTWARE para Mac OS X 10.6.3, versão 2.0: software de gestão do micro-controlador NXT e IDE da linguagem NXT-G, inspirada na linguagem LabVIEW. LEGO Group, 2008. 1 CD-ROM.

LEGO, Mindstorm NXT. Bluetooth Developer Kit. Disponível em <http://mindstorms.lego.com/en-us/support/files/default.aspx> Acesso em 19 de maio de 2010.

LEGO, Mindstorm NXT. Página de compra de sensores de toques. Disponível em <http://mindstorms.lego.com/en-us/products/default.aspx#9843>. Acesso em 6 de Junho de

2010.

LEGO, Mindstorm NXT. Página de compra de sensores de toques. Disponível em <http://mindstorms.lego.com/en-us/products/default.aspx#9846>. Acesso em 6 de Junho de 2010.

LEGO. Página oficial da Lego Groups com informações sobre o kit Lego Mindstorm NXT 2.0 antes do lançamento do mesmo. Disponível em <http://www.lego.com/eng/info/default.asp?page=pressdetail&contentid=17278&countrycode=2057&yearcode=&archive=false> Acesso em 19 de maio de 2010.

LUGER, George F. Artificial Intelligence: Structures and Strategis for Complex Problem Solving. 4. ed. Harlow, Inglaterra, 2002.

MCCARTHY, John, 1956, durante conferência realizada na Universidade de New Hampshire, EUA). Disponível em (RUSSEL, Stuart J. (Stuart Jonatahn). Inteligência Artificial: tradução da segunda edição / Stuart Russel, Peter Norvig. 2 ed. Rio de Janeiro, 2004)

MIKROKOPTER. Wiki oficial do mikrokopter. Página fornece conteúdo sobre hardware e software para a construção de vários tipos de mikrokotpr. Disponível em <http://www.mikrokopter.de/ucwiki/en/Mikrokopter-Get-started> Acessado em 8 de Junho de 2010.

NASA, página oficial do "NASA Space Telerobotics Program", fechado em 1997. Disponível em http://ranier.oact.hq.nasa.gov/telerobotics_page/telerobotics.shtm. Acesso em 6 de Junho de 2010.

PHILO. Página pessoal sobre o escritor e pesquisador Philippe E. Hurbain. Apresenta alternativa para o uso de câmeras digitais junto ao kit Lego Mindstorm NXT NXT. Disponível em < http://www.philohome.com/nxtrover/rover.htm> Acesso em 10 março 2010.

PHILO. Página pessoal sobre o escritor e pesquisador Philippe E. Hurbain. Apresenta informações sobre os cabos do kit Lego Mindstorm NXT. Disponível em <http://www.philohome.com/nxtcables/nxtcable.htm> Acesso em 19 de maio de 2010.

PHILO. Página pessoal sobre o escritor e pesquisador Philippe E. Hurbain. Apresenta

modelo de veículo construído com o kit Lego Mindstorm NXT 1.0. Disponível em <http://www.philohome.com/odin/odin.htm> Acesso em 2 abril 2010.

PHILO. Página pessoal sobre o escritor e pesquisador Philippe E. Hurbain. Apresenta testes diversos com o servo-motor do kit Lego Mindstorm NXT. Disponível em <http://www.philohome.com/nxtmotor/nxtmotor.htm> Acesso em 18 abril 2010.

RECHENBERG. Evolutionstrategie: optimierung technisher ssteme nach prinzipien der biologischen evolution. 1 ed. Frommann-Hoolzboog Verlag, 1973.

ROBOTC. Página oficial da linguagem de programação RobotC. Apresenta guia e exemplos para programação de robôs. Disponível em <http://www.robotc.net/index.php> Acesso em 07 de Maio de 2010.

RUSSEL, Stuart J. (Stuart Jonatahn). Inteligência Artificial: tradução da segunda edição / Stuart Russel, Peter Norvig. 2 ed. Rio de Janeiro, 2004.

SILVA, Leandro Nunes de Castro. Engenharia imunológica: Desenvolvimento e aplicação de ferramentas computacionais inspiradas em Sistemas Imunológicos Artificiais. UNICAMP, Campinas, 2001. Disponível em: <http://www.dca.fee.unicamp.br/~vonzuben/research/lnunes_dout/index.html>. Acesso em: 17 de maio de 2010.

SILVA, Osmar Quirino; WEIGANG, Li; YAMASHITA, Yaeko; SILVA, Paulo César Marques; MACIVER, Andrew; DENZINGER, Jörg. Otimização, Previsão e Orientação de tráfego viário utilizando Algoritmo Genético. Rio de Janeiro: XVII Congresso de Pesquisa e Ensino em Transportes, 14 de Novembro de 2003.

SPYCAMERACCTV. Loja especializada em câmeras de segurança com informações detalhadas sobre as mesmas. Disponível em <http://www.spycameracctv.com/spycamera/pinhole-wireless-nanny-hidden-camera> Acesso em 13 de maio de 2010

STEINBRUCH, Alfredo; Winterle, Paulo. Geometria analítica. 2 ed. São Paulo : Pearson Makron Books. 1987

UNIVERSIDADE DE SÃO PAULO, departamento de ciências da computação. Apresenta teoria básica sobre o Algoritmo Genético canônico. Disponível em <http://xpusp.sourceforge.net/ga_tutorial.html> Acesso em 15 de Março de 2010.

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO, grupo de teleinformática e automação. Apresenta teoria básica sobre Algoritmos Genéticos. Disponível em <http://www.gta.ufrj.br/~marcio/genetic.html> Acesso em 15 de Marco de 2010.

WINSTON, Patrick Henry. Artificial Intelligence. 3 ed. Addison Wesley. 1992

APÊNDICES

A – Código fonte da classe RobotNXT.java

```
import java.io.DataInputStream;

import java.io.DataOutputStream;

import java.io.IOException;

import lejos.nxt.Motor;

import lejos.nxt.NXT;

import lejos.nxt.SensorPort;

import lejos.nxt.UltrasonicSensor;

import lejos.nxt.comm.BTConnection;

import lejos.nxt.comm.Bluetooth;

import lejos.robotics.navigation.SimpleNavigator;


public class RobotNXT {


     @SuppressWarnings("deprecation")

     public static void main(String[] args) throws IOException,
Exception {

          final int diferencaAnguloMedicao = 10;//A diferença de
angulo de uma medição a outra

          final int numeroAngulosMedidos =
360/diferencaAnguloMedicao;//Número de angulos a ser medido a
distancia

          long [] distanciasFinal = new long
[numeroAngulosMedidos];//Variaveis de trabalho
```

```
          UltrasonicSensor ultrasonico = new
UltrasonicSensor(SensorPort.S1);//Instala sensor ultrasonico na
porta 4

          ultrasonico.ping();//Configura o sensor ultrasonico para
trabalhar no modo ping


          BTConnection btc = Bluetooth.waitForConnection();//Espera
a conexão Bluetooth


          DataInputStream dis = btc.openDataInputStream();//Canal
para chegada de dados

          DataOutputStream dos = btc.openDataOutputStream();//Canal
para saida de dados


          SimpleNavigator robo = new SimpleNavigator(32.2f,186f,
Motor.B, Motor.C);

          robo.setPosition(0,0,0);


          while(true){

               switch((int)dis.readLong()){

               case 1: //Andar para a frente ou para tras

                         {

                              long i=dis.readLong();

                              if (i<=200){

                                   Motor.B.setSpeed(180);

                                   Motor.C.setSpeed(180);

                              }else if (i<=1000){

                                   Motor.B.setSpeed(360);

                                   Motor.C.setSpeed(360);

                              }else if (i<=10000){

                                   Motor.B.setSpeed(720);
```

```
                        Motor.C.setSpeed(720);
                    }else{
                        Motor.B.setSpeed(900);
                        Motor.C.setSpeed(900);
                    }
                    robo.travel(i);
                }
                while(Motor.B.isMoving()){
                    dos.writeLong(-16);//Ainda se
movendo
                    dos.flush();
                }
                break;


            case 2: //Giro horario ou anti-horario
                    Motor.B.setSpeed(180);
                    Motor.C.setSpeed(180);
                    robo.rotate(dis.readLong());
                    while(Motor.B.isMoving() ||
Motor.C.isMoving());
                    break;

            case 3: //Ir para posição X Y
                    {
                        Motor.B.setSpeed(720);
                        Motor.C.setSpeed(720);
                        long x,y;
                        x=dis.readLong();
```

```
                                    y=dis.readLong();
                                    robo.goTo(x,y);
                                    while(Motor.B.isMoving() ||
Motor.C.isMoving());
                              }
                              break;

                  case 4: //Rotacionar para determinado angulo
                              Motor.B.setSpeed(720);
                              Motor.C.setSpeed(720);
                              robo.rotateTo(dis.readLong());
                              while(Motor.B.isMoving() ||
Motor.C.isMoving());
                              break;

                  case 5: //Leitura de distâncias
                              Motor.A.setSpeed(diferencaAnguloMedicao);
                              {
                                    int i,j;
                                    for
(i=0;i<distanciasFinal.length;i++)
                                          distanciasFinal[i]=-20;
                                    for
(i=0;i<distanciasFinal.length;i++){
      distanciasFinal[i]=ultrasonico.getDistance();
                                          Thread.sleep(300);
                                          j=0;

                                          do{
```

```
      if(distanciasFinal[i]==255){

      distanciasFinal[i]=ultrasonico.getDistance();
                                                Thread.sleep(300);
                                                j++;
                                          }else{
                                                j=3;
                                          }
                                    }while(j<3);


      Motor.A.rotate(diferencaAnguloMedicao);
                                          while(Motor.A.isMoving());
                                          {//
                                                int z;//
                                                z =
(int)dis.readLong();//

      dos.writeLong(distanciasFinal[z]*10);//
                                                dos.flush();//
                                          }//
                                    }
                                    Motor.A.setSpeed(900);

      Motor.A.rotate(diferencaAnguloMedicao*distanciasFinal.length*(-
1));
                              }
                              dis.readLong();//
                              /*{
                                    int i;
```

```
                        i = (int)dis.readLong();
                        while(i<numeroAngulosMedidos){

dos.writeLong(distanciasFinal[i]*10);
                              dos.flush();
                              i = (int)dis.readLong();
                        }
                  }*/
                  break;

            case 7: //Retorna atual heading do robô
                        dos.writeLong((long)robo.getHeading());
                        dos.flush();
                        break;
            case 8: //Retorna atual posicao X do robô
                        dos.writeLong((long)robo.getX());
                        dos.flush();
                        break;
            case 9: //Retorna atual posicao Y do robô
                        dos.writeLong((long)robo.getY());
                        dos.flush();
                        break;

            case 99://Desliga o Robo
                        dis.close();
                        dos.close();
                        btc.close();
                        NXT.shutDown();
```

```
                        break;

                default: //Erro, repita o comando
                        dos.writeLong(-19);//Repita o ultimo comando
                        dos.flush();
                        break;
                }
            }
        }
}
```

B – Código fonte da classe AlgoritmoGeneticotoNXT.java

```
import java.io.IOException;
import java.util.Random;
import java.util.BitSet;

public class AlgoritmoGeneticotoNXT {
        final int diferencaAnguloMedicao = 10;
        final int numeroAngulosLidosRobo = 360/diferencaAnguloMedicao;
        private long [] distancias = new long [numeroAngulosLidosRobo];
        private long [] fitnessNow = new long [numeroAngulosLidosRobo];
        private long [][] coordenadasMedidas = new long
[numeroAngulosLidosRobo][2];
        private long somaFitness=0,somaAngulos=0;
        private long[][] fitnessNowAngulosMedidos = new long
[2][numeroAngulosLidosRobo];//1 - fitnessNow 2 - AngulosMedidos

        LogPCtoNXT logNXT = new LogPCtoNXT();
```

```
      PCtoNXT controlador = new PCtoNXT();


      public long getSomaFitness(){

            System.out.println("Entrando no método
\"AlgoritmoGeneticotoNXT.getSomaFitness\"");

            return somaFitness;

      }

      public void setSomaFitness(long somaFitness){

            System.out.println("Entrando no método
\"AlgoritmoGeneticotoNXT.setSomaFitness\"");

            this.somaFitness = somaFitness;

      }

      public long getDistancias(int y) {

            System.out.println("Entrando no método
\"AlgoritmoGeneticotoNXT.getDistancias\"");

            return distancias[y];

      }


      public void setDistancias(long distancias, int y) {

            System.out.println("Entrando no método
\"AlgoritmoGeneticotoNXT.setDistancias\"");

            if(y>=0 && y<this.distancias.length)

                  this.distancias[y] = distancias;

      }


      public long[] getFitnessNowArray(){

            System.out.println("Entrando no método
\"AlgoritmoGeneticotoNXT.getFitnessNowArray\"");

            return fitnessNow;

      }
```

```
	public long getFitnessNow(int y) {

		System.out.println("Entrando no método
\"AlgoritmoGeneticotoNXT.getFitnessNow\"");

		return fitnessNow[y];

	}


	public void setFitnessNow(long fitnessNow,int y) {

		System.out.println("Entrando no método
\"AlgoritmoGeneticotoNXT.setFitnessNow\"");

		if(y>=0 && y<this.fitnessNow.length)

			this.fitnessNow[y] = fitnessNow;

	}

	public long getCoordenadasMedidas(int x,int y){

		System.out.println("Entrando no método
\"AlgoritmoGeneticotoNXT.getCoordenadasMedidas\"");

		return coordenadasMedidas[x][y];

	}

	public void setCoordenadasMedidas(long coordenadasMedidas,int
x,int y){

		System.out.println("Entrando no método
\"AlgoritmoGeneticotoNXT.setCoordenadasMedidas\"");

		this.coordenadasMedidas[x][y]=coordenadasMedidas;

	}


	public void movimentacao(long X4, long Y4) throws IOException{

		System.out.println("Entrando no método
\"AlgoritmoGeneticotoNXT.movimentacao\"");

		int i,j=0;

		long z1,z2;
```

```
            if(X4==Y4 && X4==999){
                  controlador.requestFimPrograma();
            }

            long distancia;
            distancia=999999999;
            for(i=(int)logNXT.getNumeroRegistro()-1;i>0;i++){
                  long distanciaTemp;
                  if(i-1==0)
                        z1=z2=0;
                  else{
                        z1=logNXT.carregaPosicaoXLidosProjeto(i-1);
                        z2=logNXT.carregaPosicaoYLidosProjeto(i-1);
                  }
                  distanciaTemp=(long)Math.pow(Math.pow(X4-
z1,2)+Math.pow(Y4-z2,2), 1/2);
                  if(distanciaTemp<distancia){
                        distancia=distanciaTemp;
                        j=i;
                  }
            }

            if(j!=0)
                  this.localizaProximoPassoDesconhecido(j);
            controlador.setProximoX(X4);
            controlador.setProximoY(Y4);
            controlador.requestCaminhoXY();
```

```
            controlador.opcao3();
      }


      public void procuraPontoDesconhecido2() throws IOException{
            System.out.println("Entrando no método
\"AlgoritmoGeneticotoNXT.procuraPontoDesconhecido\"");
            long [] distancias = new long
[this.numeroAngulosLidosRobo];
            long [] coordenadasMedidas = new long[2];//0 - x, 1 - y
            long [][] vetoresCadastrados = new long
[2][this.numeroAngulosLidosRobo];
            long angulo;
            int i,j;
            long z1,z2;

            for(i=(int)logNXT.getNumeroRegistro()-1;i>0;i++){
                  distancias=logNXT.carregaDistanciasLidasProjeto(i);
                  for(j=0;j<this.numeroAngulosLidosRobo;j++){
                        if(distancias[j]==2550){
                              if(i-1==0)

      angulo=j*this.diferencaAnguloMedicao;
                              else

      angulo=logNXT.carregaAnguloFinalProjeto(i-
1)+(j*this.diferencaAnguloMedicao);

                              do{
                                    if(angulo<0)
                                          angulo+=360;
```

```
                                    else if(angulo>=360)
                                          angulo-=360;
                                    }while(!(angulo>=0 && angulo<360));

                                    if(i-1==0)
                                          z1=z2=0;
                                    else{
      z1=logNXT.carregaPosicaoXLidosProjeto(i-1);
      z2=logNXT.carregaPosicaoYLidosProjeto(i-1);
                                    }

                                    if(angulo>=0 && angulo<90){
      coordenadasMedidas[0]=(long)(500*Math.cos(Math.toRadians(angulo
))+z1;
      coordenadasMedidas[1]=(long)(500*Math.sin(Math.toRadians(angulo
))+z2;
                                    }else if (angulo>=90 && angulo<180){
                                          coordenadasMedidas[0]=-
(long)(500*Math.cos(Math.toRadians(180-angulo)))+z1;
      coordenadasMedidas[1]=(long)(500*Math.sin(Math.toRadians(180-
angulo)))+z2;
                                    }else if(angulo>=180 && angulo<270){
      coordenadasMedidas[0]=(long)(500*Math.cos(Math.toRadians(angulo
))+z1;
      coordenadasMedidas[1]=(long)(500*Math.sin(Math.toRadians(angulo
))+z2;
```

```
                        }else{

     coordenadasMedidas[0]=(long)(500*Math.cos(Math.toRadians(-(360-
angulo))))+z1;

     coordenadasMedidas[1]=(long)(500*Math.sin(Math.toRadians(-(360-
angulo))))+z2;
                        }


                        {
                              int i1,i2;
                              long X1,X2,X3,X4,Y1,Y2,Y3,Y4;

     for(i1=(int)logNXT.getNumeroRegistro()-1;i1>=i;i1--){

     vetoresCadastrados=logNXT.carregaPontosLidosProjeto(i1);

     for(i2=0;i2<this.numeroAngulosLidosRobo;i2++){

     X1=vetoresCadastrados[i2][0];

     X2=vetoresCadastrados[i2][1];

     X3=vetoresCadastrados[(i2+1)>=this.numeroAngulosLidosRobo?i2-
this.numeroAngulosLidosRobo+1:i2+1][1];
                                          X4=coordenadasMedidas[0];

     Y1=vetoresCadastrados[i2][2];

     Y2=vetoresCadastrados[i2][3];

     Y3=vetoresCadastrados[(i2+1)>=this.numeroAngulosLidosRobo?i2-
this.numeroAngulosLidosRobo+1:i2+1][3];
                                          Y4=coordenadasMedidas[1];
```

```
                                                    if(!((X1==X2 && Y1==Y2)
|| (X1==X3 && Y1==Y3))){

     if(!this.checaSePontoEstaDentroDoTriagulo(X1, Y1, X2, Y2, X3,
Y3, X4, Y4)){

     this.localizaProximoPassoDesconhecido(i1);

     logNXT.setCoordenadasFinaisX(X4);

     logNXT.setCoordenadasFinaisY(Y4);
                                                                  return;
                                                  }
                                             }
                                        }
                                   }
                              }
                         }
                    }
          }
          System.out.println("Fim programa");
          logNXT.salvaLogProjeto();
          controlador.opcao3();
     }


     public void procuraPontoDesconhecido() throws IOException{
          System.out.println("Entrando no método
\"AlgoritmoGeneticotoNXT.procuraPontoDesconhecido\"");
          long [][] vetoresCadastrados = new long
[this.numeroAngulosLidosRobo][4];
```

```
            long[] proximaCoordenada = new long [2];//0 = x, 1 = y
            long
xMaximo,xMinimo,xTemp,yMaximo,yMinimo,yTemp,controlador;
            int i;



      for(i=1,xMaximo=yMaximo=xMinimo=yMinimo=0;i<logNXT.getNumeroReg
istro();i++){
                  xTemp=logNXT.carregaPosicaoXLidosProjeto(i);
                  if(xTemp>xMaximo)
                        xMaximo=xTemp;
                  if(xTemp<xMinimo)
                        xMinimo=xTemp;
                  yTemp=logNXT.carregaPosicaoYLidosProjeto(i);
                  if(yTemp>yMaximo)
                        yMaximo=yTemp;
                  if(yTemp<yMinimo)
                        yMinimo=yTemp;
            }


            controlador=0;


            do{


      for(proximaCoordenada[0]=xMinimo;proximaCoordenada[0]<=xMaximo;
proximaCoordenada[0]++){


      for(proximaCoordenada[1]=yMinimo;proximaCoordenada[1]<=yMaximo;
proximaCoordenada[1]++){


      for(i=1;i<logNXT.getNumeroRegistro();i++){
```

```
                                long X1,X2,X3,X4,Y1,Y2,Y3,Y4;

     vetoresCadastrados=logNXT.carregaPontosLidosProjeto(i);
                                X1=vetoresCadastrados[i][0];
                                X2=vetoresCadastrados[i][1];

     X3=vetoresCadastrados[(i+1)>=this.numeroAngulosLidosRobo?i-
this.numeroAngulosLidosRobo+1:i+1][1];
                                X4=proximaCoordenada[0];
                                Y1=vetoresCadastrados[i][2];
                                Y2=vetoresCadastrados[i][3];

     Y3=vetoresCadastrados[(i+1)>=this.numeroAngulosLidosRobo?i-
this.numeroAngulosLidosRobo+1:i+1][3];
                                Y4=proximaCoordenada[1];


                                if(!((X1==X2 && Y1==Y2) || (X1==X3
&& Y1==Y3))){

     if(!this.checaSePontoEstaDentroDoTriagulo(X1, Y1, X2, Y2, X3,
Y3, X4, Y4)){

     if(this.avaliaProximoPassoDesconhecido(X4,Y4));

     //this.localizaProximoPassoDesconhecido(X4,Y4);
                                     }
                                }
                           }
                      }
                 }
           }while(controlador!=0);//true
```

```
	}


	public boolean avaliaProximoPassoDesconhecido(long X4, long Y4)
throws IOException{
		long i,X1,Y1,numeroRegistroSelecionado=0;
		long [] distancias = new long
[this.numeroAngulosLidosRobo];
		long xDistancia,yDistancia;


		xDistancia=yDistancia=2000000000;


		for(i=0;i<logNXT.getNumeroRegistro();i++){
			if(i==1){
				X1=0;
				Y1=0;
				xDistancia=Math.abs(X4-X1);
				yDistancia=Math.abs(Y4-Y1);
				numeroRegistroSelecionado=0;
			}else{
				X1=logNXT.carregaPosicaoXLidosProjeto(i);
				Y1=logNXT.carregaPosicaoYLidosProjeto(i);
				if(Math.abs(X4-X1)<xDistancia && Math.abs(Y4-
Y1)<yDistancia){
					xDistancia=Math.abs(X4-X1);
					yDistancia=Math.abs(Y4-Y1);
					numeroRegistroSelecionado=i;
				}
			}
		}
```

```
      distancias=logNXT.carregaDistanciasLidasProjeto(numeroRegistroS
elecionado);



      if(distancias[(int)Math.toDegrees(Math.asin(Math.pow(Math.pow(x
Distancia,2)+Math.pow(yDistancia,2), 1/2)/2550))]==2550){


      this.localizaProximoPassoDesconhecido(numeroRegistroSelecionado
);
                  logNXT.setCoordenadasFinaisX(X4);
                  logNXT.setCoordenadasFinaisY(Y4);
                  this.proximoPasso();
                  return true;
            }
            else
                  return false;


      }


      public void localizaProximoPassoDesconhecido(long
numeroRegistro) throws IOException{
            long i;
            for(i=logNXT.getNumeroRegistro();i>=numeroRegistro;i--){
                  if(i==1){
                        logNXT.setCoordenadasFinaisX(0);
                        logNXT.setCoordenadasFinaisY(0);
                        this.proximoPasso();
                  }else{
```

```
     logNXT.setCoordenadasFinaisX(logNXT.carregaPosicaoXLidosProjeto
(i-1));


     logNXT.setCoordenadasFinaisY(logNXT.carregaPosicaoYLidosProjeto
(i-1));
                        this.proximoPasso();
                  }
            }
      }


      public void proximoPasso() throws IOException{
            System.out.println("Entrando no método
\"AlgoritmoGeneticotoNXT.decideProximoPasso\"");


            controlador.setProximoX(logNXT.getCoordenadasFinaisX());
            controlador.setProximoY(logNXT.getCoordenadasFinaisY());
            controlador.requestCaminhoXY();
      }


      public void comparaFitnessNowFitnessAll() throws IOException{
            System.out.println("Entrando no método
\"AlgoritmoGeneticotoNXT.comparaFitnessNowFitnessAll\"");
            long distanciaTemp,angulo;
            long [][] coordenadasMedidas = new long
[2][this.numeroAngulosLidosRobo];
            long [][] vetoresCadastrados = new long
[this.numeroAngulosLidosRobo][4];
            int i,x,z;


            for (i=0;i<this.numeroAngulosLidosRobo;i++){
```

```
      angulo=logNXT.getAnguloFaceRobo()+(i*this.diferencaAnguloMedica
o);
                  do{
                        if(angulo<0)
                              angulo+=360;
                        else if(angulo>=360)
                              angulo-=360;
                  }while(!(angulo>=0 && angulo<360));


                  if(this.getDistancias(i)!=2550)
                        distanciaTemp=1000;
                  else
                        distanciaTemp=this.getDistancias(i)-10;


                  if(angulo>=0 && angulo<90){


      coordenadasMedidas[0][i]=(long)(distanciaTemp*Math.cos(Math.toR
adians(angulo)))+logNXT.getPosicaoRoboX();


      coordenadasMedidas[1][i]=(long)(distanciaTemp*Math.sin(Math.toR
adians(angulo)))+logNXT.getPosicaoRoboY();
                  }else if (angulo>=90 && angulo<180){
                        coordenadasMedidas[0][i]=-
(long)(distanciaTemp*Math.cos(Math.toRadians(180-
angulo)))+logNXT.getPosicaoRoboX();


      coordenadasMedidas[1][i]=(long)(distanciaTemp*Math.sin(Math.toR
adians(180-angulo)))+logNXT.getPosicaoRoboY();
                  }else if(angulo>=180 && angulo<270){
```

```
      coordenadasMedidas[0][i]=(long)(distanciaTemp*Math.cos(Math.toR
adians(angulo)))+logNXT.getPosicaoRoboX();


      coordenadasMedidas[1][i]=(long)(distanciaTemp*Math.sin(Math.toR
adians(angulo)))+logNXT.getPosicaoRoboY();
                  }else{


      coordenadasMedidas[0][i]=(long)(distanciaTemp*Math.cos(Math.toR
adians(-(360-angulo))))+logNXT.getPosicaoRoboX();


      coordenadasMedidas[1][i]=(long)(distanciaTemp*Math.sin(Math.toR
adians(-(360-angulo))))+logNXT.getPosicaoRoboY();
                  }
            }


            for(i=1;i<logNXT.getNumeroRegistro();i++){


      vetoresCadastrados=logNXT.carregaPontosLidosProjeto(i);
                  //x
                  for(x=0;x<this.numeroAngulosLidosRobo;x++){
                        long X1,X2,X3,X4,Y1,Y2,Y3,Y4;
                        X1=vetoresCadastrados[x][0];
                        X2=vetoresCadastrados[x][1];


      X3=vetoresCadastrados[(x+1)>=this.numeroAngulosLidosRobo?x-
this.numeroAngulosLidosRobo+1:x+1][1];
                        Y1=vetoresCadastrados[x][2];
                        Y2=vetoresCadastrados[x][3];


      Y3=vetoresCadastrados[(x+1)>=this.numeroAngulosLidosRobo?x-
this.numeroAngulosLidosRobo+1:x+1][3];
                        for(z=0;z<this.numeroAngulosLidosRobo;z++){
```

```
                              X4=coordenadasMedidas[0][z];
                              Y4=coordenadasMedidas[1][z];

     if(this.checaSePontoEstaDentroDoTriagulo(X1,Y1,X2,Y2,X3,Y3,X4,Y
4))
                                   this.setFitnessNow(0, x);
                         }
                    }
               }
     }


     public boolean checaSePontoEstaDentroDoTriagulo(long x1,long
y1, long x2, long y2, long x3, long y3, long x4, long y4){
          System.out.println("Entrando no método
\"AlgoritmoGeneticotoNXT.checaSePontoEstaDentroDoTriangulo\"");
          double ABC,ABD,ADC,DBC;
          ABC=this.areaTriangulo(x1, x2, y1, y2, x3, y3);
          ABD=this.areaTriangulo(x1, x2, y1, y2, x4, y4);
          ADC=this.areaTriangulo(x1, x2, x4, y4, x3, y3);
          DBC=this.areaTriangulo(x4, y4, y1, y2, x3, y3);
          if(ABC==(ABD+ADC+DBC))
               return true;
          else
               return false;


     }


     public double areaTriangulo(long x1,long x2,long y1, long y2,
long x3, long y3){
```

```
            System.out.println("Entrando no método
\"AlgoritmoGeneticotoNXT.areaTriangulo\"");

            return ((x1*y2)-(x1*y3)-(y1*x2)+(y1*x3)+(x2*y3)-
(x3*y2))/2;

      }


      public long getSomaAngulos(long angulos[]){

            System.out.println("Entrando no método
\"AlgoritmoGeneticotoNXT.getSomaAngulos\"");

            int i;


      for(i=0,somaAngulos=0;i<angulos.length;somaAngulos+=angulos[i++
]);

            return somaAngulos;

      }


      public long getSomaAngulos(){

            System.out.println("Entrando no método
\"AlgoritmoGeneticotoNXT.getSomaAngulos\"");

            return somaAngulos;

      }


      public void iniciaAlgoritmoGenetico() throws IOException{

            System.out.println("Entrando no método
\"AlgoritmoGeneticotoNXT.iniciaAlgoritmoGenetico\"");

            long [] fitnessFinal = new long
[this.numeroAngulosLidosRobo];

            int i;


            if(logNXT.getNumeroRegistro()==0){

                  logNXT.iniciaProjeto(controlador.getNomeProjeto());
```

```
            }
            logNXT.resetaDadosLog();
            this.atualizaRegistroLog();
            this.transformaDistanciasAtuaisFitnessNow();

            do{
                  this.reproducaoAlgoritmoGenetico();
                  this.crossoverAlgoritmoGenetico();

                  for (i=0;i<this.numeroAngulosLidosRobo;i++){
      fitnessFinal[i]=this.fitnessNowAngulosMedidos[0][i];
                  }

                  this.getSomaFitness(fitnessFinal);
                  this.getSomaAngulos(fitnessNowAngulosMedidos[1]);

      }while(this.getSomaFitness()/this.numeroAngulosLidosRobo!=fitne
ssFinal[0] &&

      this.getSomaAngulos()/this.numeroAngulosLidosRobo!=fitnessNowAn
gulosMedidos[1][0]);

            logNXT.setFitnessFinal(fitnessFinal[0]);
            {
                  long temp;
      temp=this.fitnessNowAngulosMedidos[1][0]+logNXT.getAnguloFaceRo
bo();

                  do{
```

```
                    if(temp<0)
                        temp+=360;
                    else if(temp>=360)
                        temp-=360;
                }while(!(temp>=0 && temp<360));
                logNXT.setAnguloFinal(temp);
            }

    this.avaliaDistanciaFinal((int)this.fitnessNowAngulosMedidos[1]
[0]/this.diferencaAnguloMedicao);
            logNXT.salvaLogProjeto();
    }
            //Avalia o máximo que o robô pode andar na direção do
novo angulo proposto, sem que choque com qualquer obstáculo
            //Caso todas as distâncias ao redor do robô já sejam
conhecidas, ele procura no log algum ponto do mapa
            //não escaneado e locomove-se para lá
    public void avaliaDistanciaFinal(int i) throws IOException{
            System.out.println("Entrando no método
\"AlgoritmoGeneticotoNXT.avaliaDistanciaFinal\"");
            long distanciaTemp,distanciaMaxima;
            long coordenadaX,coordenadaY,proximoX,proximoY;;
            long [][] vetoresCadastrados = new long
[this.numeroAngulosLidosRobo][4];
            long angulo;
            int j,z;

    angulo=logNXT.getAnguloFaceRobo()+(i*this.diferencaAnguloMedica
o);
```

```
            do{
                  if(angulo<0)
                        angulo+=360;
                  else if(angulo>=360)
                        angulo-=360;
            }while(!(angulo>=0 && angulo<360));


            distanciaMaxima=this.getDistancias(i);


            if(logNXT.getNumeroRegistro()>1){
                  for
(distanciaTemp=0;distanciaTemp<=1000;distanciaTemp+=10){


                        if(angulo>=0 && angulo<90){


      coordenadaX=(long)(distanciaTemp*Math.cos(Math.toRadians(angulo
))+logNXT.getPosicaoRoboX();


      coordenadaY=(long)(distanciaTemp*Math.sin(Math.toRadians(angulo
))+logNXT.getPosicaoRoboY();
                        }else if (angulo>=90 && angulo<180){
                              coordenadaX=-
(long)(distanciaTemp*Math.cos(Math.toRadians(180-
angulo)))+logNXT.getPosicaoRoboX();


      coordenadaY=(long)(distanciaTemp*Math.sin(Math.toRadians(180-
angulo)))+logNXT.getPosicaoRoboY();


      coordenadaX=(long)(distanciaTemp*Math.cos(Math.toRadians(angulo
))+logNXT.getPosicaoRoboX();


      coordenadaY=(long)(distanciaTemp*Math.sin(Math.toRadians(angulo
))+logNXT.getPosicaoRoboY();
```

```
                    }else{

     coordenadaX=(long)(distanciaTemp*Math.cos(Math.toRadians(-(360-
angulo))))+logNXT.getPosicaoRoboX();

     coordenadaY=(long)(distanciaTemp*Math.sin(Math.toRadians(-(360-
angulo))))+logNXT.getPosicaoRoboY();
                    }



                    for(j=1;j<logNXT.getNumeroRegistro();j++){

     vetoresCadastrados=logNXT.carregaPontosLidosProjeto(j);


     for(z=0;z<this.numeroAngulosLidosRobo;z++){
                              long X1,X2,X3,X4,Y1,Y2,Y3,Y4;
                              X1=vetoresCadastrados[z][0];
                              X2=vetoresCadastrados[z][1];

     X3=vetoresCadastrados[(z+1)>=this.numeroAngulosLidosRobo?z-
this.numeroAngulosLidosRobo+1:z+1][1];
                              X4=coordenadaX;
                              Y1=vetoresCadastrados[z][2];
                              Y2=vetoresCadastrados[z][3];

     Y3=vetoresCadastrados[(z+1)>=this.numeroAngulosLidosRobo?z-
this.numeroAngulosLidosRobo+1:z+1][3];
                              Y4=coordenadaY;


                              if(!((X1==X2 && Y1==Y2) || (X1==X3
&& Y1==Y3)))
```

```
      if(this.checaSePontoEstaDentroDoTriagulo(X1,Y1,X2,Y2,X3,Y3,X4,Y
4)){

      distanciaMaxima=distanciaTemp;
                                              break;
                                        }
                                  }
                            }
                      }

      distanciaMaxima=distanciaMaxima>1000?1000:distanciaMaxima;


            if(distanciaMaxima<10 || this.getSomaFitness()==0)
                  procuraPontoDesconhecido2();
            else{
                  logNXT.setDistanciaFinal(distanciaMaxima-
200>=0?distanciaMaxima-200:distanciaMaxima);


                  distanciaMaxima=logNXT.getDistanciaFinal();



                  if(angulo>=0 && angulo<90){

      proximoX=(long)((double)distanciaMaxima*Math.cos(Math.toRadians
(angulo)))+logNXT.getPosicaoRoboX();

      proximoY=(long)((double)distanciaMaxima*Math.sin(Math.toRadians
(angulo)))+logNXT.getPosicaoRoboY();
```

```
                }else if (angulo>=90 && angulo<180){

                     proximoX=-
(long)((double)distanciaMaxima*Math.cos(Math.toRadians(180-
angulo)))+logNXT.getPosicaoRoboX();


     proximoY=(long)((double)distanciaMaxima*Math.sin(Math.toRadians
(180-angulo)))+logNXT.getPosicaoRoboY();

                }else if(angulo>=180 && angulo<270){


     proximoX=(long)((double)distanciaMaxima*Math.cos(Math.toRadians
(angulo)))+logNXT.getPosicaoRoboX();


     proximoY=(long)((double)distanciaMaxima*Math.sin(Math.toRadians
(angulo)))+logNXT.getPosicaoRoboY();

                }else{


     proximoX=(long)((double)distanciaMaxima*Math.cos(Math.toRadians
(-(360-angulo))))+logNXT.getPosicaoRoboX();


     proximoY=(long)((double)distanciaMaxima*Math.sin(Math.toRadians
(-(360-angulo))))+logNXT.getPosicaoRoboY();

                }



                logNXT.setCoordenadasFinaisX(proximoX);

                logNXT.setCoordenadasFinaisY(proximoY);

           }

     }



     public void crossoverAlgoritmoGenetico() throws IOException{

           System.out.println("Entrando no método
\"AlgoritmoGeneticotoNXT.crossoverAlgoritmoGenetico\"");

           long [] controlador = new long
[this.numeroAngulosLidosRobo];
```

```
            BitSet [] bitFitnessTemp = new BitSet[2];

            String [] stringFitness = new String
[this.numeroAngulosLidosRobo];

            int i,x,k;

            Random random = new Random();


            for (i=0; i<this.numeroAngulosLidosRobo;i++){

                  controlador[i]=i;

                  stringFitness[i] =
Long.toBinaryString(fitnessNowAngulosMedidos[0][i]);//Transforma os
Fitness em strings binarias

            }


            k=random.nextInt(stringFitness[0].length());


            for(i=0;i<(this.numeroAngulosLidosRobo%2)*2;i+=2){

                  int a,b,c;

                  System.out.println("Crossover: " + i);


                  a=random.nextInt(this.numeroAngulosLidosRobo);

                  for(c=0;c<this.numeroAngulosLidosRobo;c++){


      if(controlador[a+c>=this.numeroAngulosLidosRobo?this.numeroAngu
losLidosRobo-(a+c):a+c]<numeroAngulosLidosRobo){


      a=a+c>=this.numeroAngulosLidosRobo?this.numeroAngulosLidosRobo-
(a+c):a+c;

                              break;

                        }

                  }
```

```
                b=random.nextInt(this.numeroAngulosLidosRobo);
                for(c=0;c<this.numeroAngulosLidosRobo;c++){

     if(controlador[b+c>=this.numeroAngulosLidosRobo?this.numeroAngu
losLidosRobo-(b+c):b+c]<numeroAngulosLidosRobo && a!=b){

     b=b+c>=this.numeroAngulosLidosRobo?this.numeroAngulosLidosRobo-
(b+c):b+c;
                          break;
                     }
                }

                bitFitnessTemp[0] = new
BitSet(stringFitness[a].length());
                bitFitnessTemp[1] = new
BitSet(stringFitness[b].length());

                for(x=0;x<k;x++){
                     if(stringFitness[a].charAt(x)=='0')
                          bitFitnessTemp[0].clear(x);
                     else
                          bitFitnessTemp[0].set(x);
                     if(stringFitness[b].charAt(x)=='0')
                          bitFitnessTemp[1].clear(x);
                     else
                          bitFitnessTemp[1].set(x);
                }

                for(x=k;x<this.numeroAngulosLidosRobo;x++){
                     if(stringFitness[a].charAt(x)=='0')
```

```
                        bitFitnessTemp[1].clear(x);
                    else
                        bitFitnessTemp[1].set(x);
                    if(stringFitness[b].charAt(x)=='0')
                        bitFitnessTemp[0].clear(x);
                    else
                        bitFitnessTemp[0].set(x);
                }


                stringFitness[a]=null;
                stringFitness[a]=new
String(bitFitnessTemp[0].toString());
                stringFitness[b]=null;
                stringFitness[b]=new
String(bitFitnessTemp[1].toString());


                controlador[a]=controlador.length+1;
                controlador[b]=controlador.length+1;
            }


            //Atualiza Log
            for (i=0;i<this.numeroAngulosLidosRobo;i++){

     logNXT.setResultadoFitnessCrossover(Long.parseLong(stringFitnes
s[i],2), i);


     this.fitnessNowAngulosMedidos[0][i]=logNXT.getResultadoFitnessC
rossover(i);
            }
            logNXT.salvaLogCrossover();
```

```
	}


	//Atualiza Log com coordenadas do Robô e numero do registro

	public void atualizaRegistroLog(){

		System.out.println("Entrando no método
\"AlgoritmoGeneticotoNXT.atualizaRegistroLog\"");

		long distanciaTemp, angulo;

		int i;



	logNXT.setAnguloFaceRobo(controlador.requestHeading());//Atuali
za o Angulo para qual o Robô está direcionado


	logNXT.setNumeroRegistro(logNXT.getNumeroRegistro()+1);//Atuali
za o número do registor do log a ser gravado

		logNXT.setPosicaoRoboX(controlador.requestPosicaoX());

		logNXT.setPosicaoRoboY(controlador.requestPosicaoY());


		//Atualiza log com as distâncias lidas pelo robô

		for
(i=0;i<this.numeroAngulosLidosRobo;logNXT.setDistancias(this.getDist
ancias(i), i),i++);


		//Atualiza os vetores de distâncias lidas pelo robô

		for (i=0;i<this.numeroAngulosLidosRobo;i++){



	angulo=(logNXT.getAnguloFaceRobo()+(i*this.diferencaAnguloMedic
ao));

			do{

				if(angulo<0)
```

```
                    angulo+=360;
                else if(angulo>=360)
                    angulo-=360;
            }while(!(angulo>=0 && angulo<360));


    logNXT.setVetoresMedidosX1(logNXT.getPosicaoRoboX(),i);

    logNXT.setVetoresMedidosY1(logNXT.getPosicaoRoboY(),i);


            distanciaTemp=this.getDistancias(i);


            //Anula o armazenamento de coordenadas com
distâncias desconhecidas
            if(distanciaTemp==2550)
                distanciaTemp=0;


            if(angulo>=0 && angulo<90){

    logNXT.setVetoresMedidosX2((long)(distanciaTemp*Math.cos(Math.t
oRadians(angulo)))+logNXT.getPosicaoRoboX(),i);

    logNXT.setVetoresMedidosY2((long)(distanciaTemp*Math.sin(Math.t
oRadians(angulo)))+logNXT.getPosicaoRoboY(),i);
            }else if (angulo>=90 && angulo<180){
                logNXT.setVetoresMedidosX2(-
(long)(distanciaTemp*Math.cos(Math.toRadians(180-
angulo)))+logNXT.getPosicaoRoboX(),i);

    logNXT.setVetoresMedidosY2((long)(distanciaTemp*Math.sin(Math.t
oRadians(180-angulo)))+logNXT.getPosicaoRoboY(),i);
            }else if(angulo>=180 && angulo<270){
```

```
     logNXT.setVetoresMedidosX2((long)(distanciaTemp*Math.cos(Math.t
oRadians(angulo)))+logNXT.getPosicaoRoboX(),i);


     logNXT.setVetoresMedidosY2((long)(distanciaTemp*Math.sin(Math.t
oRadians(angulo)))+logNXT.getPosicaoRoboY(),i);

                }else{


     logNXT.setVetoresMedidosX2((long)(distanciaTemp*Math.cos(Math.t
oRadians(-(360-angulo))))+logNXT.getPosicaoRoboX(),i);


     logNXT.setVetoresMedidosY2((long)(distanciaTemp*Math.sin(Math.t
oRadians(-(360-angulo))))+logNXT.getPosicaoRoboY(),i);

                }

           }

     }



     public void resetaFitnessNow(){

           System.out.println("Entrando no método
\"AlgoritmoGeneticotoNXT.resetaFitnessNow\"");

           int i;

           for
(i=0;i<this.numeroAngulosLidosRobo;this.setFitnessNow(0,i++));

     }



     public void transformaDistanciasAtuaisFitnessNow() throws
IOException{

           System.out.println("Entrando no método
\"AlgoritmoGeneticotoNXT.transformaDistanciasAtuaisFitnessNow\"");

           int i,j;

           long [] fitnessTemp = new long
[this.numeroAngulosLidosRobo];

           this.resetaFitnessNow();
```

```
          for
(i=0;i<this.numeroAngulosLidosRobo;this.setFitnessNow((this.getDista
ncias(i)), i),i++);


          j=0;

          do{

               for (i=0;i<this.numeroAngulosLidosRobo;i++){

                    fitnessTemp[i]=this.getFitnessNow(i-
1<0?this.numeroAngulosLidosRobo-1:i-1)+

                                   this.getFitnessNow(i)+


     this.getFitnessNow(i+1>=this.numeroAngulosLidosRobo?0:i+1);

                    if(this.getFitnessNow(i)<2500)

                         fitnessTemp[i]=0;

               }

               for
(i=0;i<this.numeroAngulosLidosRobo;this.setFitnessNow(fitnessTemp[i]
,i),i++);

               j++;

          }while(j!=5);


          if(logNXT.getNumeroRegistro()>1)

               this.comparaFitnessNowFitnessAll();


          //Atualiza log com Fitness Iniciais

          for
(i=0;i<this.numeroAngulosLidosRobo;logNXT.setFitness(this.getFitness
Now(i), i),i++);

     }


     public void reproducaoAlgoritmoGenetico() throws IOException{
```

```
          System.out.println("Entrando no método
\"AlgoritmoGeneticotoNXT.reproducaoAlgoritmoGenetico\"");

          double [] porcentagemIncidenciaIndividuo = new double
[this.numeroAngulosLidosRobo];

          long somaFitness;

          int i;



     somaFitness=this.getSomaFitness(this.getFitnessNowArray());

          somaFitness=somaFitness<+1?1:somaFitness;


          for(i=0;i<this.numeroAngulosLidosRobo;i++)


     porcentagemIncidenciaIndividuo[i]=(this.getFitnessNow(i)*100)/s
omaFitness;


          //Atualiza Log com incidencia inicial de cada indíviduo

          for(i=0;i<this.numeroAngulosLidosRobo;i++)


     logNXT.setPorcentagemIncidenciaIndividuo(porcentagemIncidenciaI
ndividuo[i],i);


          //Roleta de reprodução

          this.roletaReproducao(somaFitness);

     }


     public long getSomaFitness(long [] fitnessNow){

          System.out.println("Entrando no método
\"AlgoritmoGeneticotoNXT.getSomaFitness\"");

          long somaFitness;

          int i;
```

```
            for
(i=0,somaFitness=0;i<fitnessNow.length;somaFitness+=fitnessNow[i++])
;
            this.somaFitness=somaFitness;
            return this.somaFitness;
      }


      public void roletaReproducao(long somaFitness) throws
IOException{
            System.out.println("Entrando no método
\"AlgoritmoGeneticotoNXT.roletaReproducao\"");
            long [] roletaReproducaoFitness = new long
[this.numeroAngulosLidosRobo];
            long [] roletaReproducaoAngulos = new long
[this.numeroAngulosLidosRobo];
            int i,j,k,r;


            for(i=0;i<this.numeroAngulosLidosRobo;i++){


      fitnessNowAngulosMedidos[0][i]=this.getFitnessNow(i);


      fitnessNowAngulosMedidos[1][i]=i*this.diferencaAnguloMedicao;
            }


            java.util.Random random = new java.util.Random();


            for (i=0;i<this.numeroAngulosLidosRobo;i++){


                  for (j=0;j<this.numeroAngulosLidosRobo;j++){
                        r =
random.nextInt((int)(somaFitness<+1?1:somaFitness))+1;
                        for (k=0;k<this.numeroAngulosLidosRobo;k++){
```

```
                        r-=this.getFitnessNow(k);
                        if(r<=0){
      roletaReproducaoFitness[j]=fitnessNowAngulosMedidos[0][k];
      roletaReproducaoAngulos[j]=fitnessNowAngulosMedidos[1][k];
                              k=this.numeroAngulosLidosRobo;
                        }
                  }
            }

            fitnessNowAngulosMedidos[0]=roletaReproducaoFitness;
            fitnessNowAngulosMedidos[1]=roletaReproducaoAngulos;

            //Incrementa iteracao
            logNXT.setIteracao(logNXT.getIteracao()+1);


      somaFitness=this.getSomaFitness(roletaReproducaoFitness);
         }
         //Atualiza log com os angulos e fitness escolhidos na
iteracao
      atualizaFitnessEscolhidosIteracao(fitnessNowAngulosMedidos[0]);
      atualizaAngulosEscolhidosIteracao(fitnessNowAngulosMedidos[1]);
         logNXT.salvaLogReproducao();
      }

      public void atualizaAngulosEscolhidosIteracao(long []
angulosEscolhidosIteracao){
```

```
            System.out.println("Entrando no método
\"AlgoritmoGeneticotoNXT.atualizaAngulosEscolhidosIteracao\"");

            int i;

            for (i=0;i<this.numeroAngulosLidosRobo;i++)


      logNXT.setResultadoAngulosIteracao(angulosEscolhidosIteracao[i]
, i);

      }


      public void atualizaFitnessEscolhidosIteracao(long []
fitnessEscolhidosIteracao){

            System.out.println("Entrando no método
\"AlgoritmoGeneticotoNXT.atualizaFitnessEscolhidosIteracao\"");

            int i;

            for (i=0;i<this.numeroAngulosLidosRobo;i++)


      logNXT.setResultadoFitnessIteracao(fitnessEscolhidosIteracao[i]
, i);

      }

}
```

C – Código Fonte da classe DispositivoBluetooth.java

```
import java.io.DataInputStream;

import java.io.DataOutputStream;

import java.io.IOException;

import lejos.pc.comm.NXTConnector;


public class DispositivoBluetooth {

      NXTConnector conectorBluetooth;


      DataOutputStream dos;
```

```
      DataInputStream dis;


      public void conectaBluetoothNXT(){

            System.out.println("Entrando no método
\"DispositivoBluetooth.conectaBletoothNXT\"");

            conectorBluetooth = new NXTConnector();

            boolean conectado =
conectorBluetooth.connectTo("btspp://");



            if (!conectado) {

                  System.err.println("Falha ao conectar com algum
micro-controlador NXT");

                  System.exit(1);

            }

      }


      public void desconectarBluetoothNXT(){

            System.out.println("Entrando no método
\"DispositivoBluetooth.desconectarBluetoothNXT\"");

            try {

                  dis.close();

                  dos.close();

                  conectorBluetooth.close();

            } catch (IOException ioe) {

                  System.out.println("IOException fechando conexão:");

                  System.out.println(ioe.getMessage());

            }

      }
```

```
	public void enviaMensagemBluetooth(long valor){

		System.out.println("Entrando no método
\"DispositivoBluetooth.enviaMensagemBluetooth\"");

		dos = conectorBluetooth.getDataOut();

		try {

			System.out.println("Enviando: " + valor);

			dos.writeLong((valor));

			dos.flush();


		} catch (IOException ioe) {

			System.out.println("IO Exception enviando bytes:");

			System.out.println(ioe.getMessage());

		}

	}


	public long recebeMensagemBluetooth(){

		System.out.println("Entrando no método
\"DispositivoBluetooth.recebeMensagemBluetooth\"");

		dis = conectorBluetooth.getDataIn();

		try {

			long i;

			i = dis.readLong();

			System.out.println("Recebido: " + i);

			return i;

		} catch (IOException ioe) {

			System.out.println("IO Exception recebendo bytes:");

			System.out.println(ioe.getMessage());

			return -21;

		}
```

```
    }
}
```

D – Códifo fonte da classe LogPCtoNXT.java

```
import java.io.FileNotFoundException;
import java.io.File;
import java.io.IOException;
import java.io.RandomAccessFile;

public class LogPCtoNXT {
    RandomAccessFile fileProjeto;//0
    RandomAccessFile fileReproducao;//1
    RandomAccessFile fileCrossover;//2
    String[] stringFiles = new String[3];
    final int diferencaAnguloMedicao = 10;
    final int numeroAngulosLidosRobo = 360/diferencaAnguloMedicao;
    private long numeroRegistro;
    private long posicaoRoboX;
    private long posicaoRoboY;
    private long anguloFaceRobo;
    private long iteracao;
    private long fitnessFinal;
    private long anguloFinal;
    private long distanciaFinal;
    private long[] coordenadasFinais = new long[2];//X2 e Y2 finais
    private long [] distancias = new long [numeroAngulosLidosRobo];
    private long [] fitness = new long [numeroAngulosLidosRobo];
    private double [] porcentagemIncidenciaIndividuo = new double
[numeroAngulosLidosRobo];
    private long [] resultadoAngulosIteracao = new long
[numeroAngulosLidosRobo];
    private long [] resultadoFitnessIteracao = new long
[numeroAngulosLidosRobo];
    private long [] resultadoFitnessCrossover = new long
[numeroAngulosLidosRobo];
    private long [][] vetoresMedidos = new long
[numeroAngulosLidosRobo][4];

    public LogPCtoNXT(){
          System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.LogPCtoNXT\"");
          this.setNumeroRegistro(0);
```

```
		this.setPosicaoRoboX(0);
		this.setPosicaoRoboY(0);
	}
	public void iniciaProjeto(String nomeProjeto){
		System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.iniciaProjeto\"");
		File fileTemp;
		String stringTemp;
		stringTemp =
("/Users/ivsucram/Documents/AlgoritmoGenetico/" + nomeProjeto);

		//Cria arquivo de log projeto.txt
		fileTemp = null;
		fileTemp = new File(stringTemp);
		fileTemp.mkdirs();
		fileTemp = new File(stringTemp+"/projeto.txt");
		try {
			fileTemp.createNewFile();
		} catch (IOException e) {
			System.out.println("Falha ao criar arquivo
projeto.txt");
			e.printStackTrace();
			System.exit(1);
		}
		stringFiles[0]=fileTemp.toString();

		//Cria arquivo de log reproduca.txt
		fileTemp = null;
		fileTemp = new File(stringTemp);
		fileTemp.mkdirs();
		fileTemp = new File(stringTemp+"/reproducao.txt");
		try {
			fileTemp.createNewFile();
		} catch (IOException e) {
			System.out.println("Falha ao criar arquivo
reproducao.txt");
			e.printStackTrace();
		}
		stringFiles[1]=fileTemp.toString();

		//Cria arquivo de log crossover.txt
		fileTemp = null;
		fileTemp = new File(stringTemp);
		fileTemp.mkdirs();
		fileTemp = new File(stringTemp+"/crossover.txt");
		try {
```

```
                fileTemp.createNewFile();
            } catch (IOException e) {
                System.out.println("Falha ao criar arquivo
crossover.txt");
                e.printStackTrace();
                System.exit(1);
            }
            stringFiles[2]=fileTemp.toString();
        }
        public long getCoordenadasFinaisX(){
            System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.getCoordenadasFinaisX\"");
            return coordenadasFinais[0];
        }
        public long getCoordenadasFinaisY(){
            System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.getCoordenadasFinaisY\"");
            return coordenadasFinais[1];
        }
        public void setCoordenadasFinaisX(long x){
            System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.setCoordenadasFinaisX\"");
            this.coordenadasFinais[0]=x;
        }
        public void setCoordenadasFinaisY(long y){
            System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.setCoordenadasFinaisY\"");
            this.coordenadasFinais[1]=y;
        }
        public long getFitnessFinal(){
            System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.getFitnessFinal\"");
            return fitnessFinal;
        }
        public void setFitnessFinal(long fitnessFinal){
            System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.setFitnessFinal\"");
            this.fitnessFinal=fitnessFinal;
        }
        public long getAnguloFinal(){
            System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.getAnguloFinal\"");
            return anguloFinal;
        }
        public void setAnguloFinal(long anguloFinal){
```

```
          System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.setAnguloFinal\"");
          this.anguloFinal=anguloFinal;
    }
    public long getDistanciaFinal(){
          System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.getDistanciaFinal\"");
          return distanciaFinal;
    }
    public void setDistanciaFinal(long distanciaFinal){
          System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.setDistanciaFinal\"");
          this.distanciaFinal=distanciaFinal;
    }
    public long getResultadoFitnessCrossover(int y){
          System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.getResultadoFitnessCrossover\"");
          return resultadoFitnessCrossover[y];
    }
    public void setResultadoFitnessCrossover(long
resultadoFitnessCrossover,int y){
          System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.setResultadoFitnessCrossover\"");

    this.resultadoFitnessCrossover[y]=resultadoFitnessCrossover;
    }
    public long getVetoresMedidosX1(int y){
          System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.getVetoresMedidosX1\"");
          return vetoresMedidos[y][0];
    }
    public long getVetoresMedidosX2(int y){
          System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.getVetoresMedidosX2\"");
          return vetoresMedidos[y][1];
    }
    public long getVetoresMedidosY1(int y){
          System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.getVetoresMedidosY1\"");
          return vetoresMedidos[y][2];
    }
    public long getVetoresMedidosY2(int y){
          System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.getVetoresMedidosY2\"");
          return vetoresMedidos[y][3];
    }
```

```
    public void setVetoresMedidosX1(long x1,int y){
          System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.setVetoresMedidosX1\"");
          this.vetoresMedidos[y][0]=x1;
    }
    public void setVetoresMedidosX2(long x2,int y){
          System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.setVetoresMedidosX2\"");
          this.vetoresMedidos[y][1]=x2;
    }
    public void setVetoresMedidosY1(long y1,int y){
          System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.setVetoresMedidosY1\"");
          this.vetoresMedidos[y][2]=y1;
    }
    public void setVetoresMedidosY2(long y2,int y){
          System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.setVetoresMedidosY2\"");
          this.vetoresMedidos[y][3]=y2;
    }
    public long getResultadoFitnessIteracao(int y){
          System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.getResultadoFitnessIteracao\"");
          return resultadoFitnessIteracao[y];
    }
    public void setResultadoFitnessIteracao(long
resultadoFitnessIteracao,int y){
          System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.setResultadoFitnessIteracao\"");

    this.resultadoFitnessIteracao[y]=resultadoFitnessIteracao;
    }
    public long getResultadoAngulosIteracao(int y){
          System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.getResultadoAngulosIteracao\"");
          return resultadoAngulosIteracao[y];
    }
    public void setResultadoAngulosIteracao(long
resultadoAngulosIteracao,int y){
          System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.setResultadoAngulosIteracao\"");

    this.resultadoAngulosIteracao[y]=resultadoAngulosIteracao;
    }
    public double getPorcentagemIncidenciaIndividuo(int y){
```

```
        System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.getPorcentagemIncidenciaIndividuo\"");
        return porcentagemIncidenciaIndividuo[y];
    }
    public void setPorcentagemIncidenciaIndividuo(double
porcentagemIncidenciaIndividuo,int y){
        System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.setPorcentagemIncidenciaindividuo\"");
        this.porcentagemIncidenciaIndividuo[y] =
porcentagemIncidenciaIndividuo;
    }
    public long getIteracao(){
        System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.getIteracao\"");
        return iteracao;
    }
    public void setIteracao(long iteracao){
        System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.setIteracao\"");
        this.iteracao = iteracao;
    }
    public long getNumeroRegistro(){
        System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.getNumeroRegistro\"");
        return numeroRegistro;
    }
    public void setNumeroRegistro(long numeroRegistro){
        System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.setNumeroRegistro\"");
        this.numeroRegistro = numeroRegistro;
    }
    public long getPosicaoRoboX(){
        System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.getPosicaoRoboX\"");
        return posicaoRoboX;
    }
    public void setPosicaoRoboX(long posicaoRoboX){
        System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.setPosicaoRoboX\"");
        this.posicaoRoboX = posicaoRoboX;
    }
    public long getPosicaoRoboY() {
        System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.getPosicaoRoboY\"");
        return posicaoRoboY;
    }
```

```
    public void setPosicaoRoboY(long posicaoRoboY) {
          System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.setPosicaoRoboY\"");
          this.posicaoRoboY = posicaoRoboY;
    }
    public long getAnguloFaceRobo() {
          System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.getAnguloFaceRobo\"");
          return anguloFaceRobo;
    }
    public void setAnguloFaceRobo(long anguloFaceRobo) {
          System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.setAnguloFaceRobo\"");
          this.anguloFaceRobo = anguloFaceRobo;
    }
    public long getDistancias(int y) {
          System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.getDistancias\"");
          return distancias[y];
    }
    public void setDistancias(long distancias,int y) {
          System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.setDistancias\"");
          if(y>=0 && y<this.distancias.length)
          this.distancias[y] = distancias;
    }
    public long getFitness(int y) {
          System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.getFitness\"");
          return fitness[y];
    }
    public void setFitness(long fitness,int y) {
          System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.setFitness\"");
          this.fitness[y] = fitness;
    }
    public void resetaDadosLog(){
          System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.resetaDadosLog\"");
          int i;
          this.setIteracao(1);
          this.setDistanciaFinal(0);
          this.setAnguloFinal(0);
          this.setFitnessFinal(0);
          this.setCoordenadasFinaisX(0);
          this.setCoordenadasFinaisY(0);
```

```
        for (i=0;i<this.numeroAngulosLidosRobo;i++){
              this.setDistancias(0,i);
              this.setFitness(0,i);
              this.setPorcentagemIncidenciaIndividuo(0,i);
              this.setResultadoAngulosIteracao(0,i);
              this.setResultadoFitnessIteracao(0,i);
              this.setResultadoFitnessCrossover(0,i);
              this.setVetoresMedidosX1(0,i);
              this.setVetoresMedidosX2(0,i);
              this.setVetoresMedidosY1(0,i);
              this.setVetoresMedidosY2(0,i);
        }
    }

    public void salvaLogProjeto() throws IOException{
        System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.salvaLogProjeto\"");
        //----->NumeroRegistro-CoordenadasX-CoordenadasY-
AnguloFaceRobo-DiferencaAnguloMedicao
        //---->Distancias[0]-Fitness[0]-Distancias[1]-Fitness[1]-
... -Distancias[n]-Fitness[n]
        //--->AnguloFinal-DistanciaFinal-FitnessFinal
        //-->[1],x1,x2,y1,y2-[2],x1,x2,y1,y2- ... -[n]x1,x2,y1,y2
(Vetores Medidos)
        //->CoordenadasFinais (x2,y2)
        int i;

        try {
              fileProjeto = new
RandomAccessFile(stringFiles[0],"rwd");
        } catch (FileNotFoundException e1) {
              System.out.println("Erro ao abrir arquivo
\"projeto.txt\" para escrita");
              e1.printStackTrace();
              System.exit(1);
        }

        //long sizeLogProjetoIteracao=36 +
(16*this.numeroAngulosLidosRobo) + 24 +
(36*this.numeroAngulosLidosRobo) + 16;
        fileProjeto.seek(fileProjeto.length());

        //Linha 1 - 36
        fileProjeto.writeLong(this.getNumeroRegistro());//8
        fileProjeto.writeLong(this.getPosicaoRoboX());//8
        fileProjeto.writeLong(this.getPosicaoRoboY());//8
```

```
            fileProjeto.writeLong(this.getAnguloFaceRobo());//8
            fileProjeto.writeInt(this.diferencaAnguloMedicao);//4

            //Linha 2 - (16*numeroAngulosLidosRobo)
            for(i=0;i<this.numeroAngulosLidosRobo;i++){
                  fileProjeto.writeLong(this.getDistancias(i));//8
                  fileProjeto.writeLong(this.getFitness(i));//8
            }

            //Linha 3 - 24
            fileProjeto.writeLong(this.getAnguloFinal());//8
            fileProjeto.writeLong(this.getDistanciaFinal());//8
            fileProjeto.writeLong(this.getFitnessFinal());//8

            //Linha 4 - (36*numeroAngulosLidosRobo)

      for(i=0;i<this.numeroAngulosLidosRobo;i++){//24*numeroAngulosLi
dosRobo
                  fileProjeto.writeInt(i);//4

      fileProjeto.writeLong(this.getVetoresMedidosX1(i));//8

      fileProjeto.writeLong(this.getVetoresMedidosX2(i));//8

      fileProjeto.writeLong(this.getVetoresMedidosY1(i));//8

      fileProjeto.writeLong(this.getVetoresMedidosY2(i));//8
            }

            //Linha 5 - 16
            fileProjeto.writeLong(this.getCoordenadasFinaisX());//8
            fileProjeto.writeLong(this.getCoordenadasFinaisY());//8

            fileProjeto.close();
      }

      public void salvaLogReproducao() throws IOException{
            System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.salvaLogReproducao\"");
            //---->NumeroRegistro
            //--->porcentagemIncidÃªnciaIndividuo[0]-
porcentagemIncidÃªnciaIndividuo[1]- ...  -
porcentagemIncidÃªnciaIndividuo[n]
            //
            //-->Iteracao
```

```
         //->ResultadoAnguloIteracao[0]-
ResultadoFitnessIteracao[0]-ResultadoAnguloIteracao[1]-
ResultadoFitnessIteracao[1]- ... -
ResultadoAnguloIteracao[n]ResultadoFitnessIteracao[n]
         //
         int i;

         try {
               fileReproducao = new
RandomAccessFile(stringFiles[1],"rwd");
         } catch (FileNotFoundException e1) {
               System.out.println("Arquivo \"reproducao.txt\" nÃ£o
encontrado");
               e1.printStackTrace();
               System.exit(1);
         }

         //long sizeLogReproducao = 8 +
(8*this.numeroAngulosLidosRobo) + 8 +
(16*this.numeroAngulosLidosRobo);
         fileReproducao.seek(fileReproducao.length());

         //Linha 1 - 8
         fileReproducao.writeLong(this.getNumeroRegistro());//8

         //Linha 2 - (8*this.numeroAngulosLidosRobo)
         for(i=0;i<this.numeroAngulosLidosRobo;i++){

    fileReproducao.writeDouble(this.getPorcentagemIncidenciaIndivid
uo(i));//8
         }
         fileReproducao.writeChars("\n\n");

         //
         //Linha 3 - 4
         fileReproducao.writeLong(this.getIteracao());//8

         //Linha 4 - (8*this.numeroAngulosLidosRobo)
         for(i=0;i<this.numeroAngulosLidosRobo;i++){

    fileReproducao.writeLong(this.getResultadoAngulosIteracao(i));/
/8

    fileReproducao.writeLong(this.getResultadoFitnessIteracao(i));/
/8
         }
```

```
          fileReproducao.close();
    }

    public void salvaLogCrossover() throws IOException{
          System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.salvaLogCrossover\"");
          //-->NumeroRegistro
          //->ResultadoFitnessCrossover[1]-
ResultadoFitnessCrossover[2]- ... - ResultadoFitnessCrossover[n]
          int i;

          try {
                fileCrossover = new
RandomAccessFile(stringFiles[2],"rwd");
          } catch (FileNotFoundException e) {
                System.out.println("Arquivo \"crossover.txt\" nÃ£o
encontrado");
                e.printStackTrace();
                System.exit(1);
          }

          //long sizeLogCrossover = 8 +
(4*this.numeroAngulosLidosRobo);
          fileCrossover.seek(fileCrossover.length());

          //Linha 1 - 8
          fileCrossover.writeLong(this.getNumeroRegistro());//8

          //Linha 2 - (4*this.numeroAngulosLidosRobo)
          for(i=0;i<this.numeroAngulosLidosRobo;i++){

    fileCrossover.writeLong(this.getResultadoFitnessCrossover(i));/
/8
          }

          fileCrossover.close();
    }

    public long[][] carregaPontosLidosProjeto(long numeroRegistro)
throws IOException{
          System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.carregaPontosLidosProjeto\"");
          long[][] vetoresCadastrados = new long
[this.numeroAngulosLidosRobo][4];
          int i;
```

```
         try {
               fileProjeto = new
RandomAccessFile(stringFiles[0],"rwd");
         } catch (FileNotFoundException e1) {
               System.out.println("Erro ao abrir arquivo
\"projeto.txt\" para escrita");
               e1.printStackTrace();
               System.exit(1);
         }


    fileProjeto.seek((fileProjeto.length()/(this.getNumeroRegistro(
)-1))*(numeroRegistro-1));

    fileProjeto.seek(fileProjeto.getFilePointer()+36+(16*this.numer
oAngulosLidosRobo)+24);

         for (i=0;i<this.numeroAngulosLidosRobo;i++){
               fileProjeto.readInt();
               vetoresCadastrados[i][0]=fileProjeto.readLong();//X1
               vetoresCadastrados[i][1]=fileProjeto.readLong();//X2
               vetoresCadastrados[i][2]=fileProjeto.readLong();//Y1
               vetoresCadastrados[i][3]=fileProjeto.readLong();//Y2
         }

         fileProjeto.close();

         return vetoresCadastrados;
    }

    public long carregaPosicaoXLidosProjeto(long numeroRegistro)
throws IOException{
         System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.carregaPosicaoXLidosProjeto\"");
         long x;
         try {
               fileProjeto = new
RandomAccessFile(stringFiles[0],"rwd");
         } catch (FileNotFoundException e1) {
               System.out.println("Erro ao abrir arquivo
\"projeto.txt\" para escrita");
               e1.printStackTrace();
               System.exit(1);
         }
```

```
    fileProjeto.seek((fileProjeto.length()/(this.getNumeroRegistro(
)-1))*(numeroRegistro-1));

    fileProjeto.seek(fileProjeto.getFilePointer()+36+(16*this.numer
oAngulosLidosRobo)+24+(36*this.numeroAngulosLidosRobo));

          x=fileProjeto.readLong();
          fileProjeto.close();

          return x;
    }

    public long carregaPosicaoYLidosProjeto(long numeroRegistro)
throws IOException{
          System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.carregaPosicaoYLidosProjeto\"");
          long y;

          try {
                fileProjeto = new
RandomAccessFile(stringFiles[0],"rwd");
          } catch (FileNotFoundException e1) {
                System.out.println("Erro ao abrir arquivo
\"projeto.txt\" para escrita");
                e1.printStackTrace();
                System.exit(1);
          }


    fileProjeto.seek((fileProjeto.length()/(this.getNumeroRegistro(
)-1))*(numeroRegistro-1));

    fileProjeto.seek(fileProjeto.getFilePointer()+36+(16*this.numer
oAngulosLidosRobo)+24+(36*this.numeroAngulosLidosRobo)+8);

          y = fileProjeto.readLong();
          fileProjeto.close();

          return y;
    }

    public long carregaNumeroRegistrosLog() throws IOException{
          System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.carregaNumeroRegistrosLog\"");
          long i;
```

```
         long
sizeLogProjetoIteracao=36+(16*this.numeroAngulosLidosRobo)+24+(36*
this.numeroAngulosLidosRobo)+16;

         try {
               fileProjeto = new
RandomAccessFile(stringFiles[0],"rwd");
         } catch (FileNotFoundException e1) {
               System.out.println("Erro ao abrir arquivo
\"projeto.txt\" para escrita");
               e1.printStackTrace();
               System.exit(1);
         }
         i = fileProjeto.length()/sizeLogProjetoIteracao;

         return i;
   }

   public long[] carregaDistanciasLidasProjeto(long
numeroRegistro) throws IOException{
         System.out.println("Entrando no mÃ©todo
\"LogPCtoNXT.carregaDistanciasLidasProjeto\"");
         long [] distancias = new long
[this.numeroAngulosLidosRobo];
         int i;

         try {
               fileProjeto = new
RandomAccessFile(stringFiles[0],"rwd");
         } catch (FileNotFoundException e1) {
               System.out.println("Erro ao abrir arquivo
\"projeto.txt\" para escrita");
               e1.printStackTrace();
               System.exit(1);
         }


   fileProjeto.seek((fileProjeto.length()/(this.getNumeroRegistro(
)-1))*(numeroRegistro-1));
         fileProjeto.seek(fileProjeto.getFilePointer()+36);

         for(i=0;i<this.numeroAngulosLidosRobo;i++){
               distancias[i]=fileProjeto.readLong();
               fileProjeto.readLong();
         }
```

```
          fileProjeto.close();

          return distancias;
    }

    public long carregaAnguloFinalProjeto(long numeroRegistro)
  throws IOException{
          System.out.println("Entrando no mÃ©todo
  \"LogPCtoNXT.carregaAnguloFinalProjeto\"");
          long anguloFinal;

          try {
                fileProjeto = new
  RandomAccessFile(stringFiles[0],"rwd");
          } catch (FileNotFoundException e1) {
                System.out.println("Erro ao abrir arquivo
  \"projeto.txt\" para escrita");
                e1.printStackTrace();
                System.exit(1);
          }


    fileProjeto.seek((fileProjeto.length()/(this.getNumeroRegistro(
  )-1))*(numeroRegistro-1));

    fileProjeto.seek(fileProjeto.getFilePointer()+36+(16*this.numer
  oAngulosLidosRobo));

          anguloFinal=fileProjeto.readLong();

          fileProjeto.close();

          return anguloFinal;
    }
  }
```

E – Código fonte da classe PCtoNXT.java

```
import java.io.IOException;
import java.util.Scanner;

import javax.swing.SwingUtilities;
```

```
public class PCtoNXT {

	static final int diferencaAnguloMedicao = 10;
	static final int numeroAngulosLidosRobo =
360/diferencaAnguloMedicao;
	static private long distancia,angulo,proximoX,proximoY;
	static private String nomeProjeto;
	static DispositivoBluetooth dispositivoBluetooth = new
DispositivoBluetooth();
	static AlgoritmoGeneticotoNXT algoritmoGenetico = new
AlgoritmoGeneticotoNXT();
	static TesteSwing swing = new TesteSwing();

	public static void main(String[] args) throws IOException {
		System.out.println("Entrando no método
\"PCtoNXT.main\"");
		Scanner input = new Scanner(System.in);

		System.out.printf("%s%s%s%s",
				"Escolha uma opção:\n",
				"1 - Mapear novo ambiente e salvar
arquivo;\n",
				"2 - Carregar projeto e apresentar
gráfico;\n",
				"Entre com sua opção: ");
		switch(input.nextInt()){
		case 1: opcao1();
				break;
		case 2: opcao2();
				break;
			default: System.out.printf("\n%s%s",
					"Opção não válida.",
					"Programa será fechado.");
					System.exit(0);
					break;
		}
	}
	public void opcao3() throws IOException{
		System.out.println("Entrando no método
\"PCtoNXT.opcao3\"");
		Scanner input = new Scanner(System.in);
		long X,Y;
		while(true){
			System.out.printf("\nCoordenada X: ");
			X = input.nextLong();
			System.out.printf("\nCoordanada Y: ");
```

```
                Y = input.nextLong();
                algoritmoGenetico.movimentacao(X, Y);
            }
        }

        static private void opcao2(){
            System.out.println("Entrando no método
\"PCtoNXT.opcao2\"");
            Scanner input = new Scanner(System.in);
            final String nomeProjeto;
            final int razao,preenchimento;

            System.out.printf("Digite o nome do projeto: ");
            nomeProjeto = (input.nextLine());

            System.out.printf("\nEscala (Recomendavel: 1-5): ");
            razao=input.nextInt();
            System.out.printf("\n\nApresentar área preenchida?\n" +
                        "Sim = 1\n" +
                        "Nao = 0\n" +
                        "--> ");
            preenchimento=input.nextInt();

            //Modulo de carregar projeto e iniciar objetivos
            SwingUtilities.invokeLater(new Runnable() {
             public void run() {

swing.createAndShowGUI(nomeProjeto,razao,preenchimento);
             }
           });
              System.out.println("Leia isso aqui e continue a
programar");
        }

        static private void opcao1() throws IOException{
            System.out.println("Entrando no método
\"PCtoNXT.opcao1\"");
            Scanner input = new Scanner(System.in);
            dispositivoBluetooth.conectaBluetoothNXT();

            System.out.printf("Digite o nome do projeto: ");
            setNomeProjeto(input.nextLine());

            while(true){
                requestLeituraDistancias();
                algoritmoGenetico.iniciaAlgoritmoGenetico();
```

```
                algoritmoGenetico.proximoPasso();
          }
     }

     public void requestMovimentoRobo(){
          System.out.println("Entrando no método
\"PCtoNXT.requestMovimentoRobo\"");
          dispositivoBluetooth.enviaMensagemBluetooth(1);
          dispositivoBluetooth.enviaMensagemBluetooth(distancia);
     }

     public void requestMudancaAnguloRobo(){
          System.out.println("Entrando no método
\"PCtoNXT.requestMudancaAnguloRobo\"");
          dispositivoBluetooth.enviaMensagemBluetooth(2);
          dispositivoBluetooth.enviaMensagemBluetooth(angulo);
     }

     static public void requestLeituraDistancias(){
          System.out.println("Entrando no método
\"PCtoNXT.requestLeituraDistancias\"");
          dispositivoBluetooth.enviaMensagemBluetooth(5);
          {
                int j;
                for (j=0;j<numeroAngulosLidosRobo;j++){

     dispositivoBluetooth.enviaMensagemBluetooth(j);

     algoritmoGenetico.setDistancias(dispositivoBluetooth.recebeMens
agemBluetooth(), j);
                }

     dispositivoBluetooth.enviaMensagemBluetooth(numeroAngulosLidosR
obo+1);
          }
     }

     public void requestCaminhoXY(){
          System.out.println("Entrando no método
\"PCtoNXT.requestCaminhoXY\"");
          dispositivoBluetooth.enviaMensagemBluetooth(3);
          dispositivoBluetooth.enviaMensagemBluetooth(proximoX);
          dispositivoBluetooth.enviaMensagemBluetooth(proximoY);
     }

     public void requestFimPrograma(){
```

```
            System.out.println("Entrando no método
\"PCtoNXT.requestFimPrograma\"");
            dispositivoBluetooth.enviaMensagemBluetooth(99);
            System.exit(0);
      }

      public long requestPosicaoY(){
            System.out.println("Entrando no método
\"PCtoNXT.requestPosicaoY\"");
            dispositivoBluetooth.enviaMensagemBluetooth(9);
            return dispositivoBluetooth.recebeMensagemBluetooth();
      }

      public long requestPosicaoX(){
            System.out.println("Entrando no método
\"PCtoNXT.requestPosicaoX\"");
            dispositivoBluetooth.enviaMensagemBluetooth(8);
            return dispositivoBluetooth.recebeMensagemBluetooth();
      }

      public long requestHeading(){
            System.out.println("Entrando no método
\"PCtoNXT.requestHeading\"");
            dispositivoBluetooth.enviaMensagemBluetooth(7);
            return dispositivoBluetooth.recebeMensagemBluetooth();
      }

      public void setProximoX(long x){
            System.out.println("Entrando no método
\"PCtoNXT.setProximoX\"");
            proximoX=x;
      }

      public void setProximoY(long y){
            System.out.println("Entrando no método
\"PCtoNXT.setProximoY\"");
            proximoY=y;
      }

      public void setDistancia(long x){
            System.out.println("Entrando no método
\"PCtoNXT.setDistancia\"");
            distancia=x;
      }

      public void setAngulo(long x){
```

```
            System.out.println("Entrando no método
\"PCtoNXT.setAngulo\"");
            angulo=x;
      }

      public long getDistancia(){
            System.out.println("Entrando no método
\"PCtoNXT.getDistancia\"");
            return distancia;
      }

      public long getAngulo(){
            System.out.println("Entrando no método
\"PCtoNXT.getAngulo\"");
            return angulo;
      }

      static public void setNomeProjeto(String x){
            System.out.println("Entrando no método
\"PCtoNXT.setNomeProjeto\"");
            nomeProjeto = x;
      }

      public String getNomeProjeto(){
            System.out.println("Entrando no método
\"PCtoNXT.getNomeProjeto\"");
            return nomeProjeto;
      }
}
```

F- Código fonte da classe TesteSwing.java

```
import javax.swing.JScrollPane;
import javax.swing.SwingUtilities;
import javax.swing.JFrame;
import javax.swing.JPanel;
import java.awt.Dimension;
import java.awt.Graphics;
import java.io.IOException;

public class TesteSwing {
```

```
    public void createAndShowGUI(String nomeProjeto,int razao,int
preenchimento) {
      System.out.println("Entrando no método
\"TesteSwing.createandShowGUI\"");
        System.out.println("GUI foi criado dentro do EDT? "+
                SwingUtilities.isEventDispatchThread());
        JFrame f = new JFrame("Mapeamento do projeto: " +
nomeProjeto);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JScrollPane scrollPane = new JScrollPane(new
MeuPainel(nomeProjeto,razao,preenchimento),
                JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
                JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
        f.setContentPane(scrollPane);
        f.pack();
        f.setVisible(true);
    }
}

class MeuPainel extends JPanel {

      final int diferencaAnguloMedicao = 10;
      final int numeroAngulosLidosRobo = 360/diferencaAnguloMedicao;
      int razao, preenchimento;

      public MeuPainel(String nomeProjeto,int razao,int
preenchimento){
            System.out.println("Entrando no método \"MeuPainel\"");
            logNXT.iniciaProjeto(nomeProjeto);
            this.razao = razao;
            this.preenchimento = preenchimento;
      }

      private static final long serialVersionUID = 1L;
      LogPCtoNXT logNXT = new LogPCtoNXT();

    public Dimension getPreferredSize() {
      System.out.println("Entrando no método
\"MeuPainel.getPreferredSize\"");
        return new Dimension(12000,8000);
    }

    public void paintComponent(Graphics g){
      System.out.println("Entrando no método
\"MeuPainel.paintComponent\"");
        super.paintComponent(g);
```

```
        int i,j;
        long numeroRegistrosLog=0;
        long [][] vetoresLidos = new long
[numeroAngulosLidosRobo][4];

        try {

      numeroRegistrosLog=logNXT.carregaNumeroRegistrosLog();
                  logNXT.setNumeroRegistro(numeroRegistrosLog+1);
            } catch (IOException e1) {
                  // TODO Auto-generated catch block
                  e1.printStackTrace();
            }

            g.translate(12000/2,8000/2);

            for(i=1;i<=numeroRegistrosLog;i++){
                  try {
                        vetoresLidos =
logNXT.carregaPontosLidosProjeto(i);
                  } catch (IOException e) {}

                  for(j=0;j<this.numeroAngulosLidosRobo;j++){
                        int X1,X2,X3,Y1,Y2,Y3;
                        X1=(int)vetoresLidos[j][0];
                        X2=(int)vetoresLidos[j][1];

      X3=(int)vetoresLidos[j+1>=this.numeroAngulosLidosRobo?0:j+1][1]
;
                        Y1=(int)-vetoresLidos[j][2];
                        Y2=(int)-vetoresLidos[j][3];
                        Y3=(int)-
vetoresLidos[j+1>=this.numeroAngulosLidosRobo?0:j+1][3];
                        if(preenchimento!=1){
                              g.drawLine(X1/razao, Y1/razao, X2/razao,
Y2/razao);
                              g.drawLine(X2/razao, Y2/razao, X3/razao,
Y3/razao);
                        }
                        if(preenchimento==1){
                              int [] Yvector = new int [3];
                              int [] Xvector = new int [3];
                              Xvector[0]=X1/razao;
                              Xvector[1]=X2/razao;
                              Xvector[2]=X3/razao;
                              Yvector[0]=Y1/razao;
```

```
                                Yvector[1]=Y2/razao;
                                Yvector[2]=Y3/razao;
                                g.fillPolygon(Xvector, Yvector, 3);
                            }
                        }
                    }

                System.out.println("Saindo do programa");
            }
        }
```