Dissertação

Planejamento de Trajetórias e Navegação de Robôs Móveis

Caio Júlio César do Vale Fernandes da Silva

Natal, 2016

UFRN - CT - NUPEG - Campus Universitário - CEP 59.072-970 - Natal/RN - Brasil
Fone/Fax: +55 (84) 3215-3773 - www.nupeg.ufrn.br - prhanp14@nupeg.ufrn.br

Caio Júlio César do Vale Fernandes da Silva

Planejamento de Trajetórias e Navegação de Robôs Móveis

Dissertação apresentada ao Departamento de Engenharia Mecânica da Universidade Federal do Rio Grande do Norte, para a obtenção de Título de Mestre em Ciências, na Área de Engenharia Mecânica.

Orientador: Dr. Wallace Moreira Bessa

Natal
2016

UFRN/Biblioteca Central Zila Mamede
Catalogação da Publicação na Fonte

Silva, Caio Julio Cesar do Vale Fernandes da.
Planejamento de trajetórias e navegação de robôs móveis / Caio Julio Cesar do Vale Fernandes da Silva. - Natal, 2016.
94 f. : il.

Orientador: Prof. Dr. Wallace Moreira Bessa.

Dissertação (Mestrado) - Universidade Federal do Rio Grande do Norte. Centro de Tecnologia. Programa de Pós Graduação em Engenharia Mecânica.

1. Algoritmos genéticos - Dissertação. 2. Robôs móveis - Dissertação. 3. Curvas de Bézier - Dissertação. I. Bessa, Wallace Moreira. II. Título.

RN/UF/BCZM CDU 004.8

Comissão Julgadora:

Prof. Dr.
Wallace Moreira Bessa (UFRN)
(Orientador)

Prof. Dr.
João Carlos Arantes Costa Júnior (UFRN)
(Presidente da Banca)

Prof. Dr.
Anfranserai Morais Dias (UEFS)
(Avaliador Externo)

Agradecimentos

Agradeço a Deus por tornar este trabalho possível e por abençoar a todos.

Agradeço ao meu orientador, Dr. Wallace Moreira Bessa, pelas correções e pela sua paciência.

Agradeço ao professor Dr. Adilson pela brilhante idéia sobre as curvas de Bézier, que proporcionou o seguimento deste trabalho.

Agradeço ao PRH-14 da ANP, a Petrobrás, a UFRN, instituições financiadoras da pesquisa.

Agradeço a todos que me ajudaram nos testes, Marcelo Tanaka, Josiane Fernandes, Oto Albuquerque, George Oliveira, Jorge Lima, João Deodato (Caicó), Nelson Ferreira, Yuri Paiva, com sugestões e críticas que tornaram este trabalho melhor. Um agradecimento especial a George Oliveira, Nelson Ferreira e Yuri Paiva por me ajudarem a manusear o robô, carregar caixas e organizar a sala para os testes.

Agradeço a minha mãe, que teve participação direta e indireta neste trabalho, me dando apoio ajudando a corrigir minhas falhas, além de ser responsável pelo acabamento aplicado aos obstáculos.

Agradeço a minha namorada, Thaynara Rodrigues, por toda a paciência, compreensão, carinho e críticas que recebi.

Resumo

A exploração de petróleo em profundidades elevadas requer o uso de robôs móveis para realizar operações diversas, como manutenção, montagem etc. Nesse contexto, o estudo do planejamento de trajetórias e navegação desses robôs se faz relevante, visto o grande desafio que é navegar em um ambiente que não é totalmente conhecido. Assim, este trabalho tem o objetivo de criar um algoritmo de navegação, que deve realizar o planejamento da trajetória de um robô móvel que se encontra em uma dada posição (x, y) e deve atingir a posição desejada (x_d, y_d), evitando, no entanto, a colisão com qualquer obstáculo existente no caminho. Para a geração da rota global foi utilizado um algoritmo genético (*offline*), que leva em consideração apenas as coordenadas dos pontos a serem visitados. Para desviar dos possíveis obstáculos no caminho, o robô deve gerar rotas locais baseadas nas curvas de Bézier (*online*). Na implementação do programa não há qualquer informação sobre a localização ou o formato dos obstáculos, mesmo assim, o robô deve evitar os obstáculos, baseadas nas informações dos sensores de proximidade. Esta estratégia é válida na situação em que os obstáculos são pequenos em relação as distâncias entre os pontos de visitação. Os resultados das simulações e experimentos com um robô móvel real (Robotino®) demonstraram que o robô foi capaz de realizar o percurso definido pelo algoritmo genético, desviando dos obstáculos através de curvas de Bézier e atingindo as posições desejadas dentro da margem de erro definida como aceitável. As principais contribuições deste trabalho estão nas equações utilizadas para definição dos pontos de controle no cálculo ***online*** das curvas de Bézier no planejador de rotas locais, atrelado a um planejador de rotas global, com obtenção de resultados experimentais.

Palavras-chave: Algoritmos Genéticos, Robôs Móveis, Curvas de Bézier.

Abstract

Oil exploration at great depths requires the use of mobile robots to perform various operations such as maintenance, assembly etc. In this context, the trajectory planning and navigation study of these robots is relevant, as the great challenge is to navigate in an environment that is not fully known. The main objective is to develop a navigation algorithm to plan the path of a mobile robot that is in a given position (x, y) and should reach the desired position (x_d, y_d) avoiding colision with any obstacle standing in the way. The global route was generated using a genetic algorithm, which takes into account only the coordinates of the checkpoints. The mobile robot path generation based on Bézier curves was able to dodge the possible obstacles in the way. There was no information about the obstacles's shape or location during the implementation of the program yet the robot must generate local path based on information from proximity sensors located around the robot to be able to avoid collisions. This strategy is valid in the situation where the obstacles are small relative distances between the visited sites. The results of simulations and experiments with a real mobile robot shown that the robot was able to perform the route defined by the genetic algorithm, dodging obstacles by Bezier curves and reaching the desired positions within the margin of error defined as acceptable. The main contributions of this work are the equations used to define the control points in the ***online*** calculation of Bezier curves in the planner of local routes, linked to a global route planner with obtaining experimental results.

Keywords: Genetic Algorithm, Mobile Robots, Bézier Curves.

Convenções tipográficas

O seguinte padrão de conveção tipográfica foi utilizado nesta dissertação.

Conveção tipográfica

CONVENÇÃO	DESCRIÇÃO
Itálico	Variáveis ou palavras estrangeiras
MAIÚSCULA	Siglas
Negrito	Termos importantes
`Fonte teletypefont`	Código de programa
`url verde`	Endereços eletrônicos

Lista de Figuras

Lista de Tabelas

Sumário

Prefácio

A presente dissertação está dividida em 5 capítulos.

No capítulo 1 apresenta-se o posicionamento do trabalho, faz-se um breve resumo de alguns trabalhos que abordam temas similares, a motivação, explica-se os motivos que incentivaram o estudo, os objetivos, e a organização do corpo da dissertação.

No capítulo 2 trata-se da fundamentação teórica do trabalho, onde os assuntos pertinentes à dissertação são expostos.

No capítulo 3 a metodologia do trabalho é apresentada, bem como uma descrição mais detalhada do Robotino®, robô autônomo didático da Festo. O programa proposto também é elucidado neste capítulo, as equações envolvidas no processo de implementação e a metodologia experimental.

No capítulo 4, chamado de resultados e discussões, são apresentados os resultados das simulações e dos experimentos com o Robotino®. Uma análise comparativa dos resultados obtidos, tanto numéricos quanto experimentais, é apresentada

No capítulo 5 os comentários finais e propostas de continuação do trabalho são discutidos.

Capítulo 1

Introdução

O problema de planejamento de trajetória é de grande relevância no cenário atual, pois, com a descoberta do pré-sal no Brasil, o petróleo vem sendo extraído a profundidades cada vez mais elevadas e medidas de segurança para a operação precisam ser tomadas, como por exemplo, a manutenção dos poços de extração. No entanto, essa tarefa mostra-se inviável de ser executada diretamente por seres humanos. Nesse cenário cresce a necessidade de ferramentas capazes de solucionar tais problemas. Destacamos os veículos robóticos não tripulados para a exploração submarina (ROVs, AUVs).

Em um campo submarino de exploração de petróleo o ROV deve visitar vários pontos de operação para realizar reparos, por exemplo. Assim, pode-se dizer que os pontos de operação são conhecidos, mas a constante mudança do relevo subaquático devido às correntes submarinas, além da possível presença de animais de grande porte, o caminho até o ponto de operação não é totalmente conhecido. Os robôs móveis utilizados nessas operações são caros e seu custo de utilização durante a operação também, devido à alta tecnologia envolvida nos sistemas de controle, navegação e equipamentos envolvidos. Visto esses fatores, se faz necessário encontrar formas de se diminuir o tempo de operação no fundo do mar, reduzindo custos, otimizando assim as operações por eles realizadas. Portanto, faz-se necessário um planejador de rotas globais (organizar a ordem de visitação dos pontos de operação) com recursos para desvio de obstáculos (rotas locais) para dar navegabilidade ao ROV ou AUV.

Propõe-se neste trabalho o desenvolvimento de uma estratégia para a determinação de trajetórias para um robô móvel. O planejador global de trajetórias baseia-se em um

algoritmo genético (*offline*), enquanto o planejador de rotas locais, nas curvas de Bézier (*online*). Assim, um robô que se encontra em uma dada posição (x, y), deve ser capaz de descrever um percurso até atingir a posição desejada (x_d, y_d), evitando, no entanto, a colisão com os possíveis obstáculos existentes no caminho. Esta estratégia é válida quando os obstáculos são pequenos quando comparados às distâncias entre os pontos a serem visitados.

1.1 Trabalhos relacionados

O problema de planejamento de trajetórias é abordado de várias formas, sendo resolvido por diversas ferramentas computacionais e métodos de otimização. Os algoritmos que tratam desse problema podem ser classificados em *offline* e *online* (Choset, 2001). Os algoritmos *offline* se baseiam em informaçoes prévias, onde o ambiente já é conhecido, já os algoritmos *online* não assumem completo conhecimento do ambiente e utilizam sensores em tempo real para analisá-lo. Os trabalhos mais recentes, em sua maioria, utilizam algoritmos *online*, já que assumir total conhecimento de um ambiente pode vir a ser algo não realista (Galceran e Carreras, 2013). Os trabalho apresentados a seguir estão organizados da seguinte forma: planejadores de rotas locais, planejadores de rotas globais com foco nos algoritmos genéticos, algoritmos de navegação para aplicações em robos móveis com algoritmos genéticos e curvas de Bézier.

Todas as técnicas apresentadas no próximo parágrafo se enquadram em **planejamento local de trajetórias**, e o foco principal é o desvio do obstáculo e não um percurso ótimo, assim, percebe-se o foco na abordagem *online*. Na maiorias desses trabalhos foram feitas apenas simulações.

Borenstein e Koren (1991) apresentaram uma técnica chamada *Vector Field Histogram*, a qual permite rápido desvio de obstáculo para robôs móveis. O VFH é considerado um planejador de rotas locais, e não garante ótimo global, bem como pode ficar preso em armadilhas. Hu e Brady (1994) também trataram de desvio de obstáculos em tempo real, com obstáculos estáticos, seguindo uma rota traçada previamente por um planejador global de trajetórias. A teoria da decisão foi utilizada para minimizar o risco de Bayes agindo de forma a decidir entre desvio lateral ou voltar e escolher outra rota. O algoritmo foi aplicado

em simulações e em experimentos com robô móvel. Fujimori *et al.* (1997) propuseram uma técnica de navegação local, chamada de navegação adaptativa, na qual a dinâmica do robô foi levada em consideração. Para o desvio de obstáculos o robô precisa da distância entre ele e o obstáculo em três diferentes direções. A lei de navegação é uma equação diferencial de primeira ordem e o robô atinge o objetivo sem colisão com os obstáculos mudando seu ângulo de direção na navegação. Esta técnica foi testada em simulações. O foco principal do artigo está no esclarecimento da estratégia e por isso incluiu várias restrições, como, por exemplo, a velocidade do robô, a localização e a forma dos obstáculos nas aplicações. Ma *et al.* (2001) apresentaram um método híbrido inteligente incluindo lógica fuzzy e redes neurais. A presença da lógica fuzzy, ou inferência fuzzy acelera a velocidade de aprendizado da rede neural. Este método pode ser usado para controlar um robô móvel baseado nas situações de movimento do robô em tempo real. As simulações mostraram que o método pode rapidamente mapear a relação fuzzy entre as entradas e saídas. Ferreira (2004), propôs outra técnica de desvio de obstáculos, em que o robô trabalha de forma *online*, numa abordagem reativa, em que o ângulo de desvio é calculado de forma a seguir o contorno do obstáculo. O trabalho é aplicado em um robô real no qual o algoritmo proposto é testado. Os resultados das simulações e dos experimentos com o robô foram considerados dentro dos padrões definidos como aceitáveis e as diferenças entre o ambiente virtual e o real são atribuídas aos sensores de detecção e aos erros da odometria inerentes do robô.

Em relação aos trabalhos do próximo parágrafo, os algoritmos genéticos são utilizados basicamente de forma *offline*, como **planejador global de trajetórias**, na tentativa de se encontrar o melhor percurso possível. Deve-se ainda destacar que nestes trabalho as discussões e conclusões baseiam-se apenas em resultados numéricos, oriundos de simulações computacionais.

Siciliano (2006) apresentou um planejamento de trajetórias ótimas através de algoritmos genéticos. O algoritmo genético é iniciado com a geração aleatória dos indivíduos que constituem a população inicial. Cada indivíduo representa um possível caminho. Para cada uma das trajetórias, o algoritmo verifica se houve proximidade ou impacto com os obstáculos e se o caminho gerado é o menor possível. Segundo o autor, os resultados

obtidos mostram que o algoritmo é adequado para determinar a trajetória mínima em um planejamento global, onde os obstáculos existentes são estáticos e com posição previamente conhecida. Li *et al.* (2007) apresentaram um algoritmo genético especializado para planejamento de trajetórias *online* e *offline*. O programa utiliza de informação prévia inserida na população inicial para melhorar os resultados do planejamento. Um destaque especial neste trabalho vai para a estratégia de controle por lógica Fuzzy, que é utilizada para ajustar as taxas de *crossover* e mutação de forma auto-adaptativa, com o intuito de melhorar o desempenho do algoritmo genético. Pehlivanoglu *et al.* (2007) propuseram um planejamento de trajetória para veículo aéreo não tripulado via algoritmo genético vibracional. O algoritmo genético vibracional deve encontrar o melhor caminho e o veículo aéreo deve percorrer o trajeto em um curva de Bézier, já que esses veículos não podem realizar manobras ou curvas acentuadas, devido aos limites de aceleração impostos pela estrutura. Assim, as curvas de Bézier que possuem natureza suave são adequadas para a situação. Segundo o autor, a técnica de mutação vibracional é eficiente para baixas populações, e assim elimina um dos pontos negativos do algoritmo genético, que é o custo computacional crescente com o tamanho da população, pois não e necessário usar um número grande de população. O planejamento da trajetória se dá de forma *offline* no trabalho apresentado, com simulações no Matlab.

Nos trabalhos do próximo parágrafo, são mais recentes e pode-se identificar uma maior integração entre várias ferramentas (algoritmo genético, curvas de Bézier, lógica Fuzzy, redes neurais), com foco em algoritmos de navegação para aplicação em robôs móveis. Em todas as referências consultadas, em apenas duas pode-se identificar o uso das curvas de Bézier de forma *online*. Mesmo assim, é notório que esses dois trabalhos se baseiam em simulações, faltando-lhes validação experimental.

Jolly *et al.* (2009) apresentaram um planejamento de caminho para um sistema de futebol de robôs, utilizando as curvas de Bézier. Neste trabalho foram utilizados obstáculos estáticos e móveis. A curva de Bézier com quatro pontos de controle se mostrou ótima para a solução do problema. A curva de Bézier também pode ser ajustada de forma *online*, com a localização dos obstáculos sendo estimada através de um algoritmo de posição. Neste trabalho o algoritmo foi testado em simulações. Skrjank e Klancar (2010)

apresentaram um novo método de desvio de colisão cooperativa para multiplos robôs não holonômicos, baseados nas curvas de Bézier. O caminho ótimo para cada robô é obtido através da minimização de uma função de penalidade, que leva em consideração todos os comprimentos dos caminhos sujeitos às distâncias entre os robôs (distância de segurança), e sujeitas às velocidades e acelerações (que não podem exceder o máximo permitido). O resultado do planejamento do caminho, experimentos reais e algumas ideias para trabalho futuro são discutidas. Sahingoz (2014) apresentou como criar uma trajetória adequada de vôo para sistemas multi UAV (*Unmanned Aerial Vehycle*, ou veículos aéreos não tripulados) usando algorimos genéticos, em um ambiente conhecido e a uma altitude constante. Primeiro, um caminho factível foi calculado usando um algoritmo genético paralelo, então o caminho é suavizado usando curvas de Bézier para torná-lo adequado para vôo. Resultados experimentais demonstraram que apesar do custo total (comprimento dos caminhos, por exemplo) aumentar, o tempo de cumprimento da missão é consideravelmente diminuído. Em Kala e Warwick (2014), um algoritmo genético em tempo real com curvas de Bézier para planejamento de trajetória é proposto. A principal contribuição é a integração do comportamento de seguir e ultrapassar veículos tráfego em geral como heurística para coordenação entre veículos. O planejamento em alto nível é realizado pelo algoritmo de Dijkistra que indica a rota em uma rede de estradas, e o replanejamento se dá quando a estrada é bloqueada ou um obstáculo é detectado. O algoritmo genético em tempo real seleciona os pontos de controle da curva de Bézier, e assim pode-se dizer que a curva de Bézier é calculada de forma *online*. Simulações confirmam o suscesso do algoritmo em relação ao ótimo no planejamento em alto e baixo nível, replanejamento e ultrapassagem.

Durante a pesquisa bibliográfica foram encontrados e estudados trabalhos que utilizavam algoritmos genéticos, curvas de Bézier e outras técnicas para planejamento de trajetórias de robôs móveis. Os trabalhos que mais se assemelharam com esta dissertação foram os de Sahingoz (2014), Skrjank e Klancar (2010) e Kala e Warwick (2014). Em Sahingoz (2014), a rota global é planejada por um algoritmo genético e as curvas de Bézier são utilizadas para suavizar a trajetória dos UAV's, tudo de forma *offline*. Obstáculos não foram inseridos no contexto, sendo o algoritmo testado em simulações computacionais. Em Skrjank e Klancar (2010), uma estratégia baseada nas curvas de Bézier foi utilizada

para evitar colisões entre os robôs, mas não há utilização de algoritmos genéticos, nem de planejadores de rotas globais e a posição inicial e velocidade dos robôs são conhecidas. Em Kala e Warwick (2014), um algoritmo de navegação para veículos autônomos é proposto. O planejador global de trajetórias se baseia no algoritmo de Dijkstra (*offline*) e a construção de rotas locais em um algoritmo genético em tempo real que faz a escolha dos pontos de controle da curva de Bézier (*online*). Esse último trabalho se assemelha bastante com a estrutura proposta nessa dissertação, porém em Kala e Warwick (2014) não foram realizados experimentos em robôs móveis reais ou veículos autônomos reais. Pelo fato da utilização do algoritmo genético na decisão dos pontos de controle da curva de Bézier, a chance de se encontrar curva próximas do ótimo global são maiores, entretanto devido a realização de um passo a mais para se calcular a curva de Bézier, o custo computacional da curva construída tende a ser maior. Assim, fazendo-se uma análise simples pode-se dizer que em relação ao trabalho de Kala e Warwick (2014) o algoritmo tem como pontos positivos o menor custo computacional no cálculo da curva de Bézier e a apresentação de resultados experimentais com um robô móvel real. Como pontos negativos, destacamos a falta de ajuste inteligente de forma *online* da curva de Bézier, em outras palavras, a curva pode ser recalculada de forma *online* para desviar de obstáculos, mas não foi inserido um método de cálculo de melhor contorno no processo. Vale salientar que além disso, nesta dissertação o algoritmo proposto contribui com novas equações para o cálculo dos pontos de controle das curvas de Bézier para o desvio de obstáculos

1.2 Justificativa

O problema de planejamento de trajetória pode ser abordado de modo semelhante ao problema do caixeiro viajante, pois, em alguns casos, durante a trajetória alguns pontos intermediários devem ser visitados. Esse problema é conhecido por ser NP-hard (*Non deterministic Polynomial time*), e não pode ser resolvido de forma exata em tempo polinomial (Dwivedi *et al.*, 2012). O Algoritmo Genético é um dos melhores algoritmos heurísticos que tem sido usados largamente para resolver problemas do caixeiro viajante (Ahmed, 2010). Também pode ser dito que o algoritmo genético é um método heurístico usado para melhorar o espaço de solução para o problema do caixeiro viajante e apresenta

resultados próximos do ótimo global dentro de um tempo razoável (Rao e Hedge, 2015). Assim, os algoritmos genéticos proporcionam um equilíbrio entre o aproveitamento das melhores soluções e a exploração do espaço de busca, tornando-se viável a obtenção de uma resposta razoável para o problema. No entanto, não há garantia de que o ótimo global seja encontrado. (Dwivedi *et al.*, 2012).

No planejamento da trajetória, às vezes não se dispõe de todas as informações do ambiente e alguns obstáculos podem ser encontrados na trajetória planejada. Para resolver esse problema, um planejador de rotas locais deve ser inserido no algoritmo de navegação do robô e deve trabalhar aliado aos sensores de detecção de obstáculos. Existem várias técnicas de desvio de obstáculos, como grade de certeza, *Vector Field Histogram* (VFH), Campo potencial etc. Algumas dessas técnicas funcionam *online* (reage a medida que os obstáculos são detectados) e/ou *offline* (a rota é previamente definida). Neste trabalho, o planejador de rotas locais baseia-se nas curvas de Bézier e funciona de forma *online*. As curvas de Bézier tem características mais convenientes para utilização em desvios de obstáculos do que as técnicas de interpolação. Isso porque a curva de Bézier passa pelo primeiro e o último ponto de controle. Então o planejador de rotas tem total controle sobre a forma da curva de uma maneira previsível apenas mudando poucos parâmetros simples. Entretanto, a maioria das técnicas de definição de curvas usadas atualmente no planejamento de trajetórias para robôs envolvem a interpolação de um dado conjunto de pontos e a curva produzida passa por todos os pontos de controle. Além disso, tem poucos pontos de guinada (Jolly *et al.*, 2009), e a curvatura varia suavemente do ponto inicial ao ponto final, devido às derivadas continuas de alta ordem (Skrjank e Klancar, 2010).

1.3 Objetivos

Criar um algoritmo genético para o planejamento de trajetória atrelado a um algoritmo de navegação. Assim, o programa implementado pode ser dividido em dois módulos distintos: **algoritmo genético** e **algoritmo de navegação**. Os algoritmos genéticos geram a ordem em que os pontos de interesse devem ser visitados, e o algoritmo de navegação gerencia a execução da movimentação do robô, desviando dos possíveis obstáculos através de curvas de Bézier. É necessário realizar o ajuste dos parâmetros do algoritmo genético

(população, taxa de mutação, taxa de cruzamento, número de iterações) para gerar rotas mais próximas do ótimo global além do ajuste dos parâmetros do algoritmo de navegação (limite de tensão, número de pontos, peso e distância) para garantir o desvio dos obstáculos.

Capítulo 2

Fundamentação Teórica

Neste capítulo são apresentados aspectos gerais da teoria, bem como parte da nomenclatura empregada, de modo facilitar a compreensão da estratégia adotada na presente dissertação. Como a principal aplicação dos algoritmos de navegação se dá em robôs móveis, este capítulo se inicia com uma breve introdução sobre esse tipo de robô.

Os robôs móveis são dispositivos de transporte automático, ou seja, são plataformas mecânicas dotadas de um sistema de locomoção, capazes de navegar através de um determinado ambiente de trabalho, dotados de certo nível de autonomia para sua locomoção, portando cargas (Secchi, 2008).

A robótica móvel tem sido um tema frequente tanto no meio industrial quanto no meio acadêmico (Oliveira *et al.*, 2012). São exemplos de aplicação, veículos AGV (Veículos Autonomamente Guiados) em indústrias com plantas autônomas de fabricação, veículos móveis autônomos em serviços domésticos (domótica) (Nascimento, 2009), veículos aéreos não tripulados em aplicações militares, veículos subaquáticos autônomos (Kwon *et al.*, 2006), dentre outros.

Os robôs móveis podem ser classificados nas seguintes modalidades: terrestres, aéreos (*Unmanned Aerial Vehicles ou UAV's*) e subaquáticos (*Remotely Operated Vehicles ou ROV's* e *Autonomous Underwater Vehicles ou AUV's*), micro e nano-robôs. Na tabela 2.1 apresenta-se um detalhamento maior desta classificação (Secchi, 2008).

Tabela 2.1: classificação dos robôs móveis em relação ao tipo de locomoção

Classificação	Exemplos
Terrestres	Robôs com rodas
	Robôs com esteira
	Robôs com patas
	Sem forma definida
Aéreos	Drones
	Quadrirotores
	Helimodelos robotizados
	UAV's
Sub-aquáticos	ROV's
	AUV's
Outros	Micro e nanorobôs

2.1 Algoritmos Genéticos

Conforme Sivanandam e Deepa (2008), a computação evolutiva foi introduzida por volta de 1960 por Rechenberg no trabalho intitulado "*Evolution strategies*" (Estratégias de evolução). Os Algoritmos Genéticos foram propostos por John Holland e desenvolvidos em seu livro "*Adaptation in natural and artificial systems*" no ano de 1975. Neste livro ele descreve como aplicar os princípios da evolução natural em problemas de otimização, e construiu o primeiro algoritmo genético, inspirado na teoria da evolução natural na origem das espécies de Charles Darwin. Holland propôs os Algoritmos Genéticos como um método heurístico baseado na "sobrevivência do mais apto".

Os algoritmos genéticos como um método computacional inteligente é uma técnica de busca usada na ciência da computação para achar soluções aproximadas para problemas de otimização combinatorial (Dwivedi *et al.*, 2012). São implementados como uma simulação de computador em que uma população de representações abstratas de solução é selecionada em busca de soluções melhores. A evolução geralmente se inicia a partir de um conjunto de soluções criado aleatoriamente (população inicial) e é realizada por meio de gerações. A cada geração, a adaptação de cada solução na população é avaliada, alguns indivíduos são selecionados para a próxima geração, e recombinados ou mutados para formar uma nova população que é avaliada de acordo com os critérios de parada, se a solução for aceitável, o algoritmo retorna o conjunto de respostas encontrado, se não, uma nova população então é utilizada como entrada para a próxima iteração do algoritmo. Na figura 2.1 é apresentado

um exemplo de fluxograma do algoritmos genéticos.

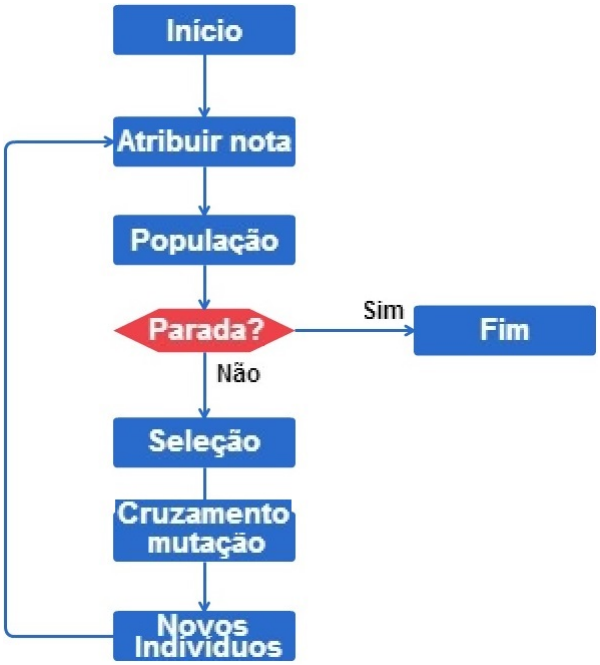

Figura 2.1: Algoritmo genético

Termos originados na biologia serviram de inspiração para a terminologia dos algoritmos genéticos. De acordo com Von Zuben (2011), cromossomos são usualmente implementados na forma de listas de atributos ou vetores, onde cada atributo é conhecido como gene. Os possíveis valores que um determinado gene pode assumir são denominados alelos. Na tabela 2.2, adaptada de Oliveira *et al.* (2012), pode-se ver mais alguns exemplos da comparação terminológica entre os termos da biologia e os algoritmos genéticos.

Tabela 2.2: Terminologia biologia x algoritmos genéticos (Tabela adaptada de Oliveira (2007))

Biologia	**Algoritmos Genéticos**
Cromossomo	Indivíduo, *string*, cromossomo, árvore
Gene	Características
Alelo	Valor
Locus	Posição
Genótipo	Estrutura
Fenótipo	Conjunto de parâmetros
População	Conjunto de pontos (indivíduos) no espaço de busca
Geração	Iteração completa do algoritmo genético

Algoritmos genéticos diferem dos algoritmos tradicionais de otimização em basicamente quatro aspectos: baseiam-se em uma codificação do conjunto das soluções possíveis, e não nos parâmetros da otimização em si; os resultados são apresentados como uma população

de soluções e não como uma solução única; não necessitam de um modelo matemático ou conhecimento das derivadas, ou de uma descrição matemática aprofundada do problema, apenas de uma forma de avaliação do resultado; usam transições probabilísticas e não regras determinísticas (Goldberg, 1989).

A seguir os principais termos e operações dos algoritmos genéticos serão descritos.

2.1.1 Indivíduos

Um indivíduo é uma solução simples. Nos algoritmos genéticos, indivíduos podem receber codificação binária, inteira, de ponto flutuante ou qualquer outro tipo que possa melhor representar o problema. Não é a codificação das variáveis o responsável maior pelo sucesso dos algoritmos genéticos, mas a codificação mais apropriada é sempre desejável (Avila, 2002).

2.1.2 População

Segundo Cordeiro (2008), uma população representa o conjunto atual de indivíduos encontrados em uma determinada iteração do algoritmo. A idéia é que de acordo com as iterações, soluções mais adequadas devam ser encontradas. De acordo com Sivanandam e Deepa (2008) idealmente a população inicial deve conter uma grande variedade de material genético de forma que todo o espaço de soluções possíveis seja explorado. A população pode ser iniciada aleatoriamente ou usar algum tipo de heurística para direcionar a busca, entretanto vale salientar que pode-se acabar restringindo o espaço de busca, fazendo com que a busca por uma solução ótima global seja dificultada. Com uma população pequena, o desempenho pode cair, pois, desse modo, a população fornece uma pequena cobertura do espaço de busca do problema. Uma grande população geralmente fornece uma cobertura representativa do domínio do problema. No entanto, para se trabalhar com grandes populações, são necessários maiores recursos computacionais (Catarina e Bach, 2003).

2.1.3 Aptidão (*Fitness*)

A aptidão corresponde a nota associada a um indivíduo ou cromossomo que permite a avaliação de quão boa é a solução por ele representada (Mitchell, 1998). A função aptidão deve ser planejada para cada problema a ser resolvido. Dado um cromossomo em particular, a função de aptidão irá retornar um simples dado numérico, que indica a utilidade ou habilidade do indivíduo representado pelo cromossomo. A aptidão é utilizada como parâmetro para operações como a de seleção.

2.1.4 Seleção

A seleção determina quais indivíduos da população irão ter todo ou parte de seu material genético transferido para a próxima geração de indivíduos, de tal forma que dê maior chance de reprodução àqueles mais adaptados ao meio ambiente, isto é, àqueles que apresentam melhor aptidão (Catarina e Bach, 2003). O objetivo do método de seleção aplicado ao algoritmo é fazer com que o material genético de boa qualidade aumente de geração em geração, enquanto que o material genético ruim venha a desaparecer ou ficar em número reduzido. Inspirado no processo de seleção natural dos seres vivos, o algoritmo seleciona os melhores indivíduos (maior aptidão) para gerar cromossomos filhos por meio de *crossover* e mutação, sempre com o objetivo de levar o algoritmo para as melhores regiões do espaço de busca. Exemplos de alguns tipos de seleção utilizados são: seleção roda de roleta (*Roulette Wheel Selection*), seleção por ranqueamento (*Rank Selection*) e a seleção por regime permanente (*Steady State Selection*).

Seleção Roda de Roleta (*Roulette Wheel Selection*)

É um dos métodos de seleção mais tradicionais nos Algoritmos Genéticos, sendo o princípio de funcionamento simples de ser explicado e implementado. Para cada indivíduo é atribuído um espaço da roleta com o tamanho proporcional ao valor da aptidão do indivíduo (Mitchell, 1998). A roda girará N vezes, onde N é o número de indivíduos da população, assim, em cada volta, um indivíduo é selecionado para fazer parte dos pais para a próxima geração.

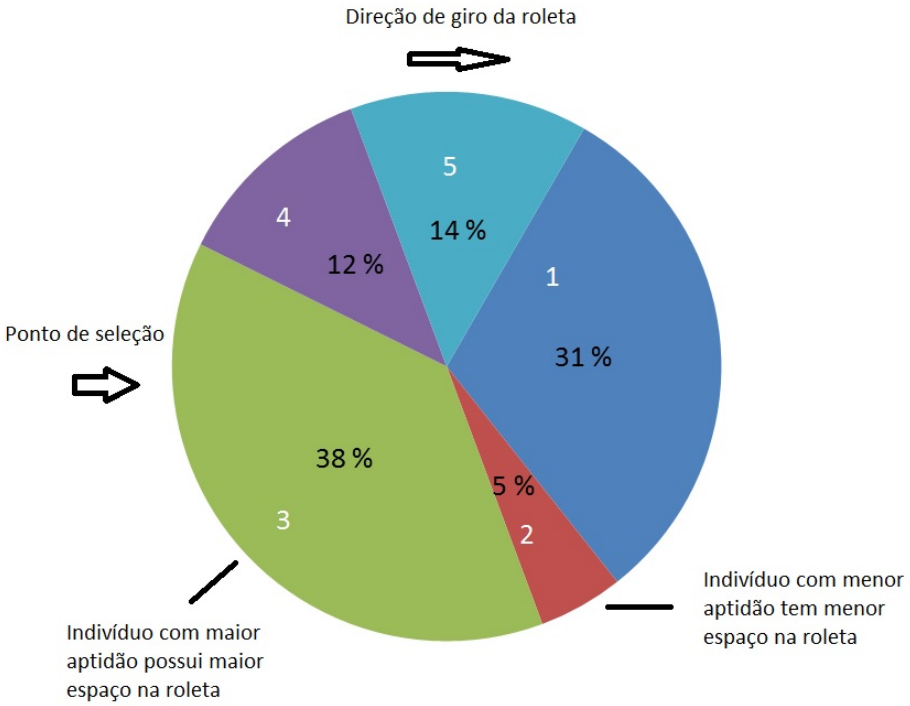

Figura 2.2: Roda de Roleta

Pode-se ver na figura 2.2 que o indivíduo de número 3 apresenta o maior valor de aptidão e portanto o maior espaço na roleta, configurando, então, maior chance de ser selecionado.

Seleção por Ranqueamento Linear (*Linear Rank Selection*)

O método de seleção roda de roleta terá problemas quando os valores das aptidões variarem muito. Os indivíduos são rankeados de acordo com o valor de aptidão, no qual aqueles com maiores aptidões ficarão com um rank mais alto e aqueles com aptidões mais baixas ficarão em posições mais baixas. Os indivíduos são então selecionados com uma probabilidade que é linearmente proporcional ao rank dos indivíduos da população (Sivaraj e Ravichandran, 2011). Este método aumenta as chances de indivíduos com baixa aptidão, trazendo para a população a ser selecionada uma maior quantidade de material genético, portanto aumentando o espaço de busca das possíveis respostas e assim também diminuindo as chances de se ficar preso em ótimos locais. Porém, a convergência pode ficar mais lenta.

Seleção por Regime Permanente (*Steady State Selection*)

É um método de seleção em que apenas poucos indivíduos são substituídos a cada geração. Esses indivíduos substituídos, são, geralmente, os indivíduos com menor aptidão, e os indivíduos que os substituem são os filhos dos indivíduos com maior aptidão da geração anterior. Algoritmos baseados em seleção por regime permanente são normalmente utilizados em sistemas baseados em evolução, nos quais aprendizagem incremental é importante (Mitchell, 1998).

2.1.5 Cruzamento (*Crossover*)

Segundo Von Zuben (2011), "O operador de *crossover* ou recombinação cria novos indivíduos através da combinação de dois ou mais indivíduos. A idéia intuitiva por trás do operador de *crossover* é a troca de informação entre diferentes soluções candidatas". Para a aplicação do *crossover*, os pais são selecionados e, através de uma probabilidade de ocorrência, o operador age formulando a troca de informações formando os novos indivíduos. Geralmente a probabilidade de ocorrência da operação é alta, acima de 50%. Quando o operador atua, cada novo indivíduo (filho) recebe características de ambos (ou todos) os pais envolvidos, quando não, os filhos são exatamente iguais aos pais. Existem vários tipos de operadores, dentre eles destacamos o *crossover* de um ponto, que é o mais popular. De acordo com Lima (2008) "Trata-se de uma recombinação entre dois cromossomos pais que trocam partes de sua cadeia binária a partir de um ponto aleatório de corte".

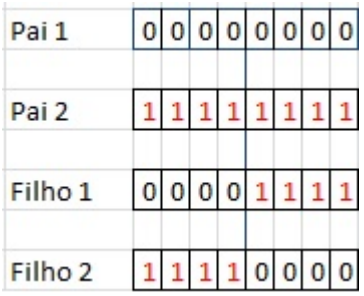

Figura 2.3: *Crossover* de um ponto

Cruzamento PMX (*Partially Mapped Crossover*)

De acordo com Saraiva e Oliveira (2010), o operador PMX foi proposto por Goldberg e Lingle (1985) para o Problema do Caixeiro Viajante. Dados dois cromossomos pais pai_1 e pai_2, dois pontos de corte são escolhidos aleatoriamente em ambos uniformemente. Estas subcadeias geradas serão o material genético de troca, sendo herdadas pelos filhos $filho_1$ e $filho_2$. Para evitar rotas inviáveis, um mapeamento é feito a fim de respeitar a restrição. Geralmente a restrição imposta para o problema do caixeiro viajante é a de que não se deve visitar uma cidade mais de uma vez, ou seja, em uma rota um número não pode se repetir. Portanto, se na troca de informação uma rota inválida for criada, os números fora da subcadeia gerada pelos pontos de corte devem ser arrumados de forma a gerar uma rota válida.

$$\begin{aligned} pai_1 &= (1,2|3,4,5,|6) \\ pai_2 &= (6,3,|1,4,5,|2) \\ &\downarrow \\ filho_1 &= (3,2,|1,4,5,|6) \\ filho_2 &= (6,1,|3,4,5,|2) \end{aligned} \tag{2.1}$$

Como pode ser visto na equação 2.1 (adaptada de Silva (2006)), as cidades 3, 4 e 5 do pai_1 e 1, 4, e 5 do pai_2, formam o material genético de troca. Porém, verifica-se que nesta troca, uma rota inválida seria gerada, pois o pai_1 já tem a cidade 1 em sua rota e o pai_2 também já tem a cidade 3. Para resolver esse problema, o mapeamento 1-3, 4-4,5-5 é realizado, e, no pai_1, onde havia 1 fora da região de troca foi-se substituído por 3, finalmente dando origem ao $filho_1$, e, no pai_2, onde havia 3 fora da região de troca foi-se substituído por 1, dando origem ao $filho_2$.

2.1.6 Mutação

Segundo Cordeiro (2008), esta estratégia consiste em mutar aleatoriamente um indivíduo existente de forma que um novo indivíduo seja criado. O objetivo da mutação é promover a variabilidade do material genético, ou seja, é um mecanismo que serve para

ajudar a evitar que o algoritmo fique preso em um mínimo local. A operação de mutação ocorre com probabilidades bem menores do que a de crossover, ficando geralmente por volta de 1% de chance de ocorrência.

2.2 Curvas de Bézier

As curvas de Bézier foram desenvolvidas por Pierre Bézier, funcionário da Renault, sendo publicadas pela primeira vez em 1962, para o design de automóveis. Hoje em dia as curvas de Bézier são largamente utilizadas na computação gráfica e animação (Choi *et al.*, 2010), usinagem (figura 2.4, Piratebrine (2013)) (Souza, 2001), e tem propriedades úteis para o problema de geração de caminhos (Choi *et al.* (2010), Skrjank e Klancar (2010), Jolly *et al.* (2009), dentre outros).

Figura 2.4: Curva de Bézier aplicada a usinagem (Piratebrine, 2013)

Pontos de controle (figura 2.5) são utilizados para definir o formato da curva, onde os pontos inicial e final da curva coincidem com o primeiro e último ponto de controle, respectivamente. Sendo assim, a curva não passa pelos outros pontos de controle, ficando contida no fecho convexo (*convex hull*) dos pontos de controle, curvas de ordem elevadas podem ser geradas aumentando o número de pontos de controle na curva (Sahingoz, 2014). Quanto maior o número de pontos de controle, maior a instabilidade numérica, em que a mudança de um simples ponto pode mudar o formato da curva (Weisstein, 2009). As

curvas podem ser transladadas e rotacionadas através de transformações geométricas nos pontos de controle (Weisstein, 2009).

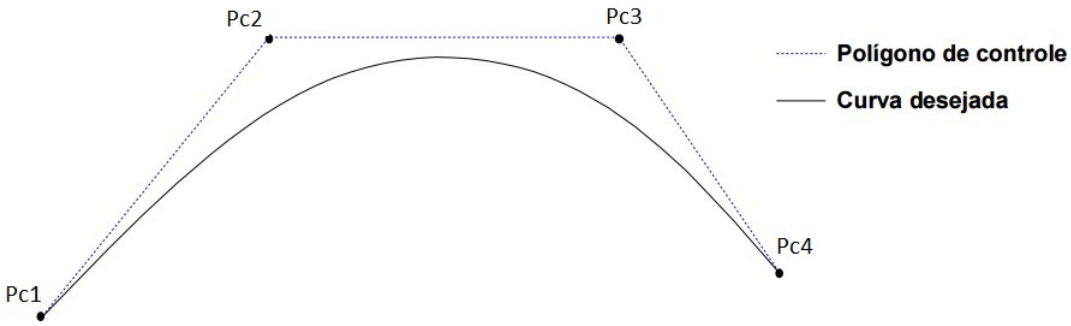

Figura 2.5: Pontos de controle e curva de Bézier (Souza, 2001)

As curvas de Bézier tem características mais convenientes para utilização em desvios de obstáculos do que as técnicas de interpolação. Isso porque a curva de Bézier passa pelo primeiro e o último ponto de controle. Então o planejador de rotas tem total controle sobre a forma da curva de uma maneira previsível apenas mudando poucos parâmetros simples. Entretanto, a maioria das técnicas de definição de curvas usadas atualmente no planejamento de trajetórias para robôs envolvem a interpolação de um dado conjunto de pontos e a curva produzida passa por todos os pontos de controle. As curvas de bézier tem poucos pontos de guinada, são mais suaves que as splines cúbicas (Jolly *et al.*, 2009), e a curvatura da curva varia suavemente do ponto inicial ao ponto final, devido às derivadas continuas de alta ordem (Skrjank e Klancar, 2010).

A seguinte definição matemática foi adaptada de Jolly *et al.* (2009).

Uma curva de Bézier paramétrica é definida por:

$$P(u) = \sum_{i=0}^{n} Pc_i J_{n,i}(u), \qquad 0 \leq u \leq 1 \tag{2.2}$$

Onde $P(u)$ representa o conjunto de pontos da curva, u um parâmetro que varia entre [0,1], n o grau da base de Bernstein e Pc_i os pontos de controle da curva de Bézier.

Para uma curva de grau n, $(n+1)$ pontos de controle são requeridos. A função de Bernstein é dada por:

$$J_{n,i}(u) = {}^{n}C_i u^i (1-u)^{n-i} \tag{2.3}$$

onde:

$$^{n}C_i = \frac{n!}{(n-i)!i!}$$

Uma curva de Bézier de terceiro grau pode ser definida por quatro pontos de controle. Sejam $Pc_0 = (A_0, B_0)$, $Pc_1 = (A_1, B_1)$, $Pc_2 = (A_2, B_2)$, $Pc_3 = (A_3, B_3)$ os pares coordenados dos pontos de controle, nos quais A_i está para as coordenadas do eixo x e B_i para as coordenadas do eixo y, assim:

$$\begin{aligned} P_x(u) &= \sum_{i=0}^{3} A_i J_{n,i}(u) \\ &= A_0(1-u)^3 + 3A_1u(1-u)^2 + 3A_2u^2(1-u) + A_3u^3 \qquad (2.4) \\ P_y(u) &= \sum_{i=0}^{3} B_i J_{n,i}(u) \\ &= B_0(1-u)^3 + 3B_1u(1-u)^2 + 3B_2u^2(1-u) + B_3u^3 \qquad (2.5) \end{aligned}$$

As equações 2.4 e 2.5 podem ser expandidas e rearranjadas em um polinômio de terceira ordem em u, assim:

$$\begin{aligned} P_x(u) &= a_0 + a_1u + a_2u^2 + a_3u^3 \qquad (2.6) \\ P_y(u) &= b_0 + b_1u + b_2u^2 + b_3u^3 \qquad (2.7) \end{aligned}$$

onde a_0, a_1, a_2, a_3 e b_0, b_1, b_2, b_3 são coeficientes definidos em termos das componentes dos pontos de controles.

As curvas de Bézier foram usadas nesta dissertação para que no momento em que o obstáculo for detectado uma curva seja gerada de modo a contorná-lo e evitar a colisão.Observa-se, portanto, que além das características já citadas anteriormente no texto, as curvas de Bézier são facilmente calculáveis e de fácil implementação.

A seguir, os materiais e métodos são discutidos, com descrição mais focada no robô utilizado (Robotino®), nas equações implementadas e no programa proposto.

Capítulo 3

Materiais e métodos

Neste capítulo, os materiais, o robô utilizado, as equações envolvidas e a metodologia experimental são explicadas em detalhes.

3.1 Materiais

Os materiais utilizados no trabalho são listados e descritos nessa seção, sendo divididos em recursos físicos e recursos computacionais.

Recursos Físicos	**Recursos Computacionais**
Computador	*Microsoft Visual Studio* 2012
Obstáculos	Linguagem C++
Fita adesiva	API 1.0 para o Robotino®
Trena e paquímetro	Robotino *SIM Demo*
Robotino®	Gnuplot
Câmeras	Windows 10 64 bits

Tabela 3.1: Recursos Físicos e Computacionais

3.1.1 Recursos Físicos

O computador utilizado tem as seguintes características:

Memória RAM	8.00 GB
Processador	Intel Core i7-4700MQ 2.40GHz
Disco Rígido	1 TB
Placa de Vídeo	Nvidia GeForce GTX 765M

Tabela 3.2: Características do computador

A trena (Robust ref. 73-5) foi utilizada para fazer as marcações dos pontos de visitação e o paquímetro utilizado para medição dos erros de posição do robô e na medição do tamanho dos obstáculos. As características do paquímetro estão descritas na tabela 3.3.

Faixa	Resolução	Exatidão	Número Categoria
150mm/6”	0.05mm e 1/128”	± 0.05mm e ±0.002”	125MEB-6/150

Tabela 3.3: Características do Paquímetro. Exatidão conforme norma NBR NM 216:2000.

Os obstáculos (figura 3.1) foram construídos de forma se parecer com os obstáculos presentes no simulador (figura 3.2). No geral os obstáculos tem forma cilíndrica, e foram adaptados a partir de potes de suplemento alimentar. Existem 4 diferentes tamanhos, sendo cada um representado por uma letra: A, B, C ou D, cujas medidas são apresentadas na tabela 3.4. Os obstáculos A, B e C são únicos, entretanto que existem 6 obstáculos D, que foram colocados juntos, pois eram pequenos e geralmente não eram detectados devido aos pontos cegos dos sensores de detecção de obstáculos.

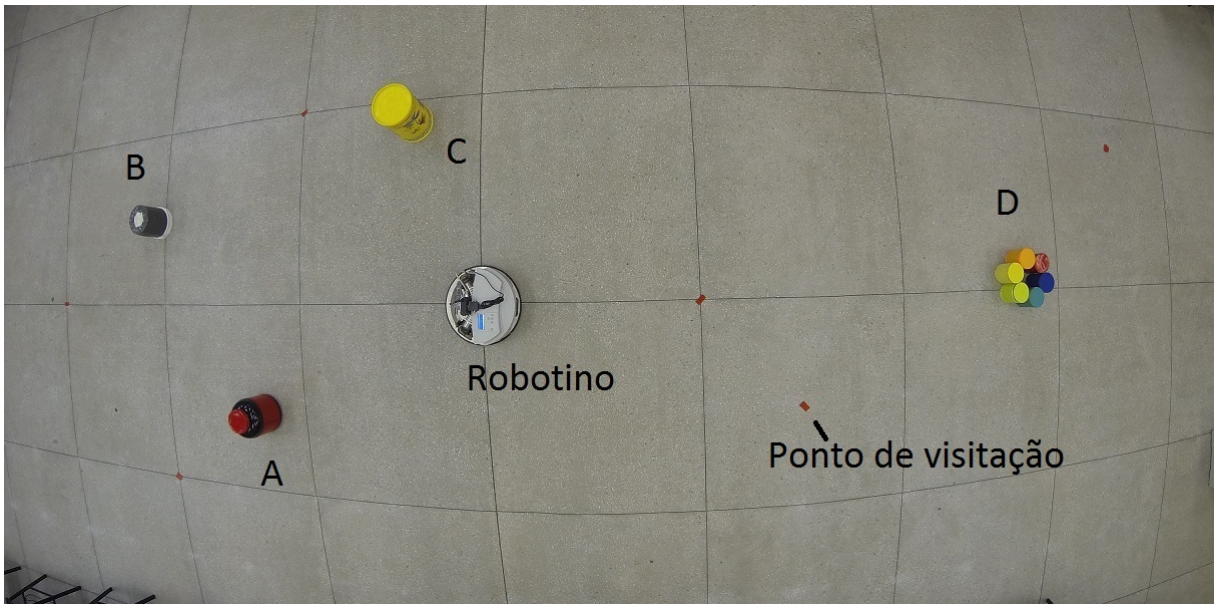

Figura 3.1: Esquema do ambiente do experimento

Obstáculo	A	B	C	D
Diâmetro	204.5 mm	161.3 mm	200.0 mm	104.5 mm
Altura	259.3 mm	188.2 mm	301.8 mm	128.2 mm

Tabela 3.4: Medidas dos obstáculos.

A fita adesiva vermelha foi usada para ajudar a fazer as marcações dos pontos de visitação e da posição do Robotino® (3.1).

O Robotino® é um robô móvel didático da Festo® e foi o robô utilizado para a aplicação do sofware desenvolvido neste trabalho. Devido à importância do robô, uma seção posterior será apresentada para explicar mais detalhes do equipamento.

As câmeras foram utilizadas para registros em fotos e vídeos do trabalho apresentado.

3.1.2 Recursos Computacionais

A API utilizada foi a 1.0, para o C++, que pode ser baixada no endereço openrobotino.org, que também apresenta toda a documentação das classes, funções e variáveis. Esta API já veio instalada no Robotino®, e portanto para manter um padrão entre as simulações foi utilizada a mesma versão nas simulações no computador

A linguagem de programação C++ foi utilizada pra escrever o código do programa, pois proporciona flexibilidade e velocidade de comunicação na API 1.0.

A IDE utilizada neste trabalho foi o Microsoft® Visual Studio 2012. A API foi construída na época para o Microsoft® Visual Studio 2010, mas a versão de 2012 também se mostrou compatível.

O simulador é disponibilizado pela Festo® , e é chamado de Robotino® SIM Demo. O simulador não necessita de um computador de configurações avançadas, porém ele pode fazer uso do recurso PhysX® das placas de video da Nvidia®, que melhora o desempenho da simulação.

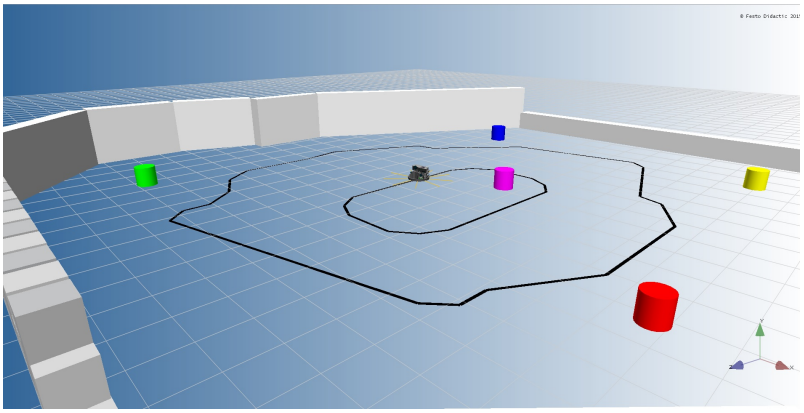

Figura 3.2: Visão do ambiente do simulador

Na próxima seção o robô utilizado nesta dissertação é apresentado, e seus principais componentes são descritos.

3.2 Robotino®

O Robotino® (figura 3.3) é um robô móvel didático da Festo®, que integra tecnologias de acionamento elétrico, cinemática, sensores, processamento de imagens e técnicas de programação. É um robô móvel terrestre **omnidirecional**, capacitado para andar em ambientes planos (2D). O termo omnidirecional é utilizado para descrever a habilidade de se mover instantaneamente para qualquer posição a partir de qualquer configuração (Dorofei *et al.*, 2007). Isso aumenta a capacidade de se mover em ambientes com obstáculos, pois pode manobrar em pequenos espaços, sem necessidade de reorientação do corpo do robô.

Figura 3.3: Robotino® do laboratório de manufatura da UFRN

Dimensões	Valores
Diâmetro	370 mm
Altura	210 mm (sem câmera)
Peso	aproximadamente 11 kg

Tabela 3.5: Dimensões do Robotino

As rodas são do tipo Mecanum, dispostas em 120^o uma da outras, contendo cada uma delas uma unidade motora individualmente controlável. Esse tipo de roda tem como pontos positivos design compacto e alta capacidade de carga, e, como pontos negativos, sensibilidade a irregularidades no piso, design da roda complexo e contato descontínuo da roda (Dorofei *et al.*, 2007). Cada unidade motora é composta pelos seguintes componentes:

1. Motor DC

2. Encoder incremental

3. Roda omnidirecional Mecanum

4. Redutor com relação de transmissão de 16:1

5. Correia dentada

Sensores infravermelho estão acoplados ao chassi, dipostos a 40^o de cada um, totalizando 9 ao todo. Eles servem para detectar obstáculos entre 4 centímetros e 30 centímetros de distância do robô. Também, em torno do chassi, encontra-se o sensor anti-colisão do parachoque, o qual para o robô no momento em que colide com algo que ofereça resistência ao movimento (figura 3.4).

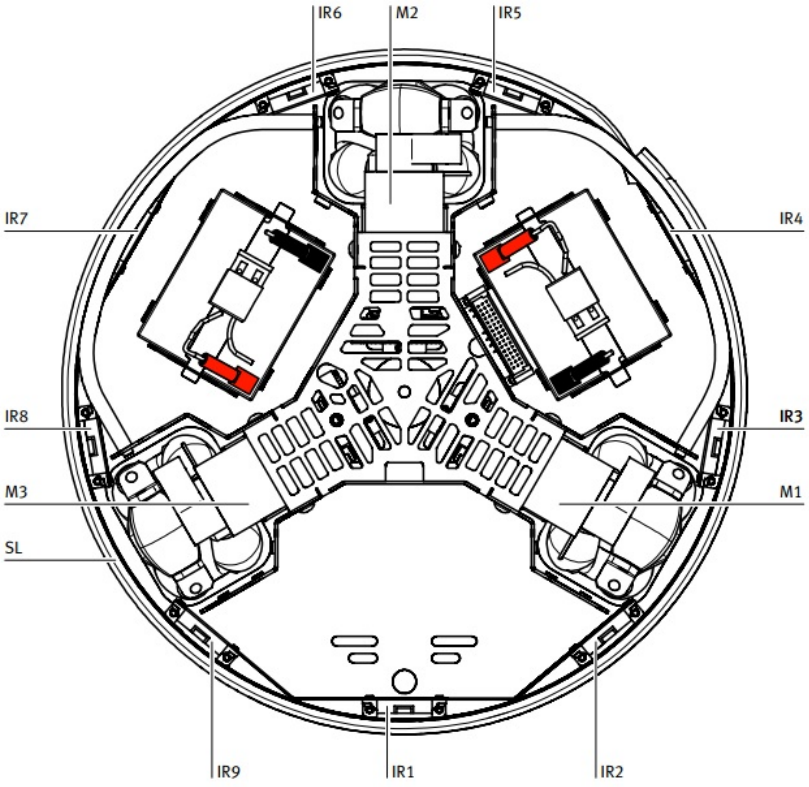

Figura 3.4: Vista de cima. IR1 a IR9 - sensores infravermelhos de 1 a 9. M1 a M3 - Motores de 1 a 3. SL - sensor anti colisão (Weber e Bellenberg, 2010).

O Robotino® possui ainda duas entradas USB, uma Ethernet, oito entradas analógicas, oito entradas digitais, oito saídas digitais, dois relés para atuadores adicionais, entrada para cartão de memória.

A unidade de controle consiste de 3 componentes:

1. procesador PC 104, compatível com MOPSlcdVE, 300 MHz, sistema operacional Linux com *real-time* kernel, SDRAM de 128 MB.

2. *Compact flash card* com API (*Aplication Programming Interface*) (Interface pela qual o programa de aplicação acessa o sistema operacional e outros serviços) em C++ para controlar o Robotino®

3. Ponto de acesso Wireless LAN.

O giroscópio (figura 3.5) é necessário para executar movimentos de rotação precisos, sendo de extrema importância no caso de longas distâncias percorridas. Depois de que esse acessório é instalado, a API do Robotino® faz o uso de seus dados automaticamente.

Figura 3.5: Giroscópio (Festo, 2013b)

3.3 Visão geral do programa e equações implementadas

Os algoritmos foram testados no simulador, e depois aplicados na plataforma experimental (Robotino®).

O robô deverá visitar um conjunto de pontos de interesse, que servirão como base para o traçado da trajetória pelo algoritmo genético, ou seja, o usuário deve inserir os pontos de visitação no programa, em qualquer ordem (exceto o ponto inicial e final), e então o algoritmo genético se encarregará de traçar o melhor caminho. Assim, as coordenadas dos pontos de visitação devem ser conhecidas. O caminho entre os pontos de visitação é desconhecido, e portanto o algoritmo de navegação deve ser capaz de desviar de possíveis

obstáculos, mas para que funcione corretamente, não pode haver obstáculos no ponto de interesse.

Os pontos de visitação são definidos de acordo com o problema do usuário do programa. Nesta dissertação os pontos foram decididos de forma arbitrária, de forma que o robô executasse circuitos ou percursos de forma a explorar os quatro quadrantes do espaço bidimensional.

Todo o processo de geração da rota global é feito pelo algoritmo genético, ocorre de forma *offline* e acontece antes da navegação, assim, primeiro será descrito o algoritmo genético utilizado e depois o algoritmo de navegação que funciona de forma *online*.

3.3.1 Algoritmo Genético

O algoritmo genético deve organizar a ordem dos pontos de visitação de forma *offline* e depois passar essas informações para o algoritmo de navegação. A seguir, os parâmetros e métodos utilizados para construir o algoritmo genético são explicados.

Indivíduo

Codificação da população foi decidida como inteira, seguindo o exemplo de autores como Sivanandam e Deepa (2008), Sahingoz (2014). Portanto, cada indivíduo (cromossomo) é considerado como uma rota, sendo cada ponto de visitação representado por um número inteiro diferente, com a restrição de que para ser considerado como uma resposta de fato, os pontos de visitação não se repitam dentro de uma mesma rota (excetuando-se o ponto de retorno). O tamanho do cromossomo depende do número de pontos de visitação. Na tabela 3.6, encontra-se um exemplo de rota válida e de rota inválida, para uma população de tamanho 2 (duas rotas), cromossomo de tamanho 8 (vetor de tamanho 8) e sete pontos de visitação.

Rota 1	0	5	3	2	4	1	6	0	$\longrightarrow$ Rota válida
Rota 2	0	3	2	2	1	1	6	0	$\longrightarrow$ Rota inválida

Tabela 3.6: Rotas válida e inválida

População

A primeira população foi gerada de forma aleatória, sem informação prévia das distâncias ou coordenadas envolvidas, com a restrição de que os locais de visitação dentro de uma rota não podem se repetir. Depois de um certo número de iterações a população deve evoluir e encontrar respostas mais adequadas ao problema. A quantidade de iterações e o tamanho da população foram definidos na seção dos resultados, de forma empírica.

Aptidão (*Fitness*)

As coordenadas em duas dimensões são inseridas no programa e então o cálculo das distâncias das rotas (possíveis respostas) é realizado. Para o problema em questão, quanto menor for a rota, maior a aptidão do indivíduo, ou seja, quanto menor a soma dos trechos da distância euclidiana ponto a ponto da configuração de pontos apresentada, maior a chance do indivíduo ser escolhido para as operações genéticas (*crossover*, mutação). Assim, tomando como exemplo a rota 1 da tabela 3.6:

$$D_{rota_1} = \sqrt{(x_5 - x_0)^2 + (y_5 - y_0)^2 + (x_3 - x_5)^2 + (y_3 - y_5)^2 + (x_2 - x_3)^2 + (y_2 - y_3)^2 + \sqrt{+(x_4 - x_2)^2 + (y_4 - y_2)^2 + (x_1 - x_4)^2 + (y_1 - y_4)^2 + \sqrt{+(x_6 - x_1)^2 + (y_6 - y_1)^2 + (x_0 - x_6)^2 + (y_0 - y_6)^2}}} \quad (3.1)$$

onde D_{rota_i} representa a distância percorrida pela rota i, x_i a coordenada do ponto i na direção X, y_i a coordenada do ponto i na direção Y.

A menor rota é então escolhida por comparação com as outras.

Seleção

O método de seleção escolhido foi o de regime permanente (*steady state*), conforme proposto por Sivanandam e Deepa (2008). São selecionados dois pais, considerados os melhores indivíduos da geração, ou seja com a maior aptidão atual (menor distância). Esse tipo de seleção é adequado nos casos onde deve-se adquirir novas informações, mantendo-se algumas outras importantes. Ou seja, criar novas rotas sem esquecer bons caminhos já aprendidos anteriormente.

Crossover

Como indicado por vários autores (Sivanandam e Deepa, 2008; Von Zuben, 2011; Linden, 2008), o operador de *crossover* selecionado foi o PMX (*partially mapped crossover*), pois esse operador evita que a prole (novas rotas) gerada possa ter problemas como pontos de operação repetidos (visitados mais de uma vez em uma mesma rota) e não visitados, o que invalidaria a rota. Então, dependendo da taxa de ocorrência de crossover a operação ocorrerá ou não. Essa taxa é determinada no capítulo 4.

Mutação

A mutação também poderá ocorrer aleatóriamente, com uma taxa bem menor que a taxa de ocorrência de crossover, com o objetivo de impedir que o algoritmo fique preso em ótimos locais. A mutação ocorre da seguinte forma, uma das posições do cromossomo é escolhida e tem seu valor alterado aleatoriamente, na faixa de número de pontos de visitação. Como não pode haver dois números iguais no mesmo cromossomo (rota inválida 3.6), é feito uma troca de posições caso isso ocorra. A taxa de mutacão controla a ocorrência da operação. O ajuste da taxa é realizada no capítulo 4.

Critério de parada

O critério de parada adotado foi o de número máximo de iterações, visto que não há informações sobre o ótimo global para se fazer comparações. Outro critério pensado foi o de diminuição da distância em relação à distância inicial, por exemplo, se o melhor indivíduo da geração atual tiver sua rota reduzida em 50% em relação a inicial então essa será a resposta. Porém, se o chute inicial for muito bom, provavelmente esse valor nunca será alcançado. Portanto, por questões de simplicidade de implementação, o número máximo de iterações foi selecionado. O número de iterações do algoritmo é definido no capítulo 4.

3.3.2 Algoritmo de Navegação

Com a rota gerada e escolhida, o robô recebe então as informações para a navegação. O robô deve seguir em linha reta do ponto em que está até o objetivo. Se o cromossomo tiver

tamanho 8, o robô tem 8 objetivos para completar (8 pontos de visitação). O primeiro objetivo é completado automaticamente (ponto de partida, origem), somente indo em direção ao próximo ponto se passar pelo objetivo atual com uma tolerância admissível (tabela 3.7). Com valores maiores do que os da tabela 3.7, o robô ficou visivelmente mais longe do objetivo original, e com valores menores o robô não conseguia atingir a condição admissível para então passar para o próximo objetivo, devido aos valores pequenos e a velocidade determindada. Assim, para a odometria, a tolerância admissível é mostrada na tabela 3.7.

Tole X sensor	20 mm
Tole Y sensor	20 mm

Tabela 3.7: Tabela de tolerância admissível

Nesta estratégia, o Robotino® utiliza os sensores internos para sua localização e orientação, ou seja um sistema de odometria baseado nas medições do encoder digital e no giroscópio. O método da odometria proporciona uma boa exatidão a curtas distâncias, mas a medida que distâncias maiores são percorridas, os erros de posição também crescem, pois a integração incremental ao longo do tempo acumula erros. Para um melhor desempenho na estimativa da posição do robô é aconselhável utilizar medidas de posicionamento absolutas.

Posição relativa

Assim que os objetivos são organizados pelos algoritmos genéticos, o robô começa a andar em direção ao próximo objetivo assim como também calcula sua posição relativa ao próximo objetivo. A posição relativa é dada por:

$$Cd_x = obj_x - odometry_x \tag{3.2}$$

$$Cd_y = obj_y - odometry_y \tag{3.3}$$

onde Cd_x representa a posição relativa ao objetivo na direção X, Cd_y a posição relativa ao objetivo na direção Y, obj_x o objetivo na direção X, obj_y o objetivo na direção Y, $odometry_x$ a posição atual do Robotino® na direção X, $odometry_y$ a $\rightarrow$ posição atual do Robotino® na direção Y.

Durante o percusso o robô analisa se atingiu ou não o objetivo dentro de uma margem de erro aceitável. O erro é calculado da seguite forma:

$$erro_x = |Cd_x| \tag{3.4}$$

$$erro_y = |Cd_y| \tag{3.5}$$

onde $erro_x$ representa a diferença ente a posição atual na direção X e o objetivo na direção X, $erro_y$ representa a diferença ente a posição atual na direção Y e o objetivo na direção Y

Assim, se ambos $erro_x$ e $erro_y$ retornarem um valor entre 0 e 20 mm, uma função controla o Robotino® para tomar novas novas decisões (parar ou seguir para outro objetivo). Estes valores foram definidos a partir de observações no simulador e no robô real.

Para que essa estratégia funcione, o sistema de referência do robô deve coincidir com o sistema fixo em relação ao ambiente.

Orientação

A orientação em relação ao objetivo é calculada utilizando-se uma função interna do C++, chamada de `atan2(y,x)`, a qual calcula o ângulo referente a tangente formada por $\frac{Cd_y}{Cd_x}$, levando em consideração o quadrante de acordo com as coordenadas. A orientação do robô móvel (figura 3.6) em relação ao seu próprio eixo, é realizada controlando-se a velocidade individual de cada motor. Isto é feito a partir de uma função da API do Robotino® chamada de `motor.setSpeedSetPoint(valor)`. Se todos os 3 motores tiverem o mesmo valor positivo, o Robotino® gira no sentido antihorário (aumenta o ângulo ϕ). Se todos tiverem o mesmo valor negativo, o Robotino® gira no sentido horário (diminui o ângulo ϕ). A função controla a orientação do Robotino® para que este se mantenha entre $-5^o \leq \phi \leq 5^o$ em relação ao seu próprio eixo.

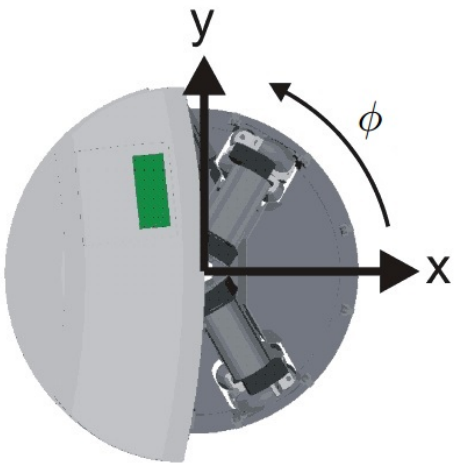

Figura 3.6: Orientação do Robotino® (Festo, 2013c).

Velocidade

A velocidade é calculada de forma que sua resultante sempre seja um valor fixo, que foi escolhida como 0.2 m/s, pois foi observado que o impacto de uma batida a essa velocidade no robô ainda era aceitável. Acima desse valor de velocidade poderiam haver problemas, sendo isso baseado em observações empíricas, além de que se fosse preciso pará-lo manualmente essa velocidade (0.2 m/s) seria admissível. O Robotino ® utiliza como padrão para velocidade a unidade de milímetros por segundo, portanto para o cálculo no programa as unidades precisam ser convertidas.

O robô deve se mover aproximadamente em linha reta se nenhum obstáculo for detectado, assim temos as seguintes equações que regem o controle da velocidade do robô:

$$vel_r = 200 \tag{3.6}$$

$$ang = arctg(Cd_y/Cd_x) \tag{3.7}$$

$$vel_x = vel_r cos(ang) \tag{3.8}$$

$$vel_y = vel_r sin(ang) \tag{3.9}$$

onde vel_r representa o módulo da velocidade resultante (mm/s), ang a direção da posição relativa ao objetivo, $arctg$ o arco tangente, vel_x a velocidade em X, vel_y a velocidade em Y, $cos(ang)$ o cosseno de ang, $sin(ang)$ o seno de ang.

Distância ao obstáculo

Como no cálculo da rota nenhuma informação sobre obstáculos foi inserida, o robô deve desviar dos possíveis obstáculos a partir das informações sensoriais recebidas durante a navegação. Para tanto, os sensores utilizados foram os 9 sensores infravermelhos que ficam ao redor do chassi do Robotino® (figura 3.4).

Os sensores infravemelhos, da forma que são equipados no robô, apresentam pontos cegos. Na figura 3.7, é demonstrado como os sensores detectam os obstáculos. As estreitas faixas amarelas ao redor do robô indicam a faixa na qual os sensores começam a detectar obstáculos, tudo que não é indicado pela faixas amarelas são pontos cegos.

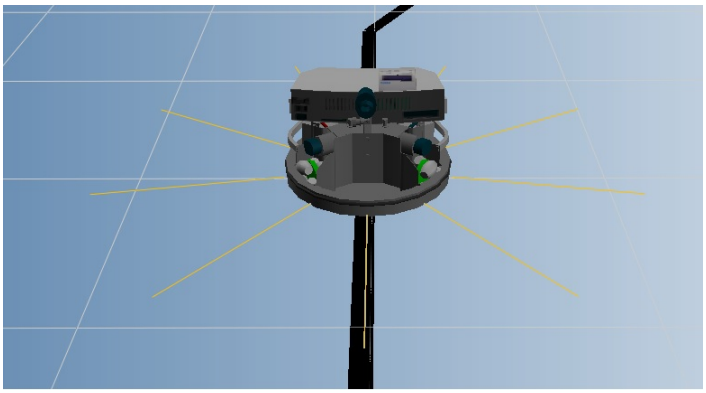

Figura 3.7: Pontos cegos

Um obstáculo é encontrado quando o valor de tensão referente a um dos sensores fica acima de um certo valor, que foi definido experimentalmente, e o ajuste desse valor de tensão é apresentado em detalhes no capítulo 4. Os sensores infravermelhos tem o comportamento apresentado na figura 3.8, com um range de aproximadamente 40 cm e uma zona morta de aproximadamente 4 cm.

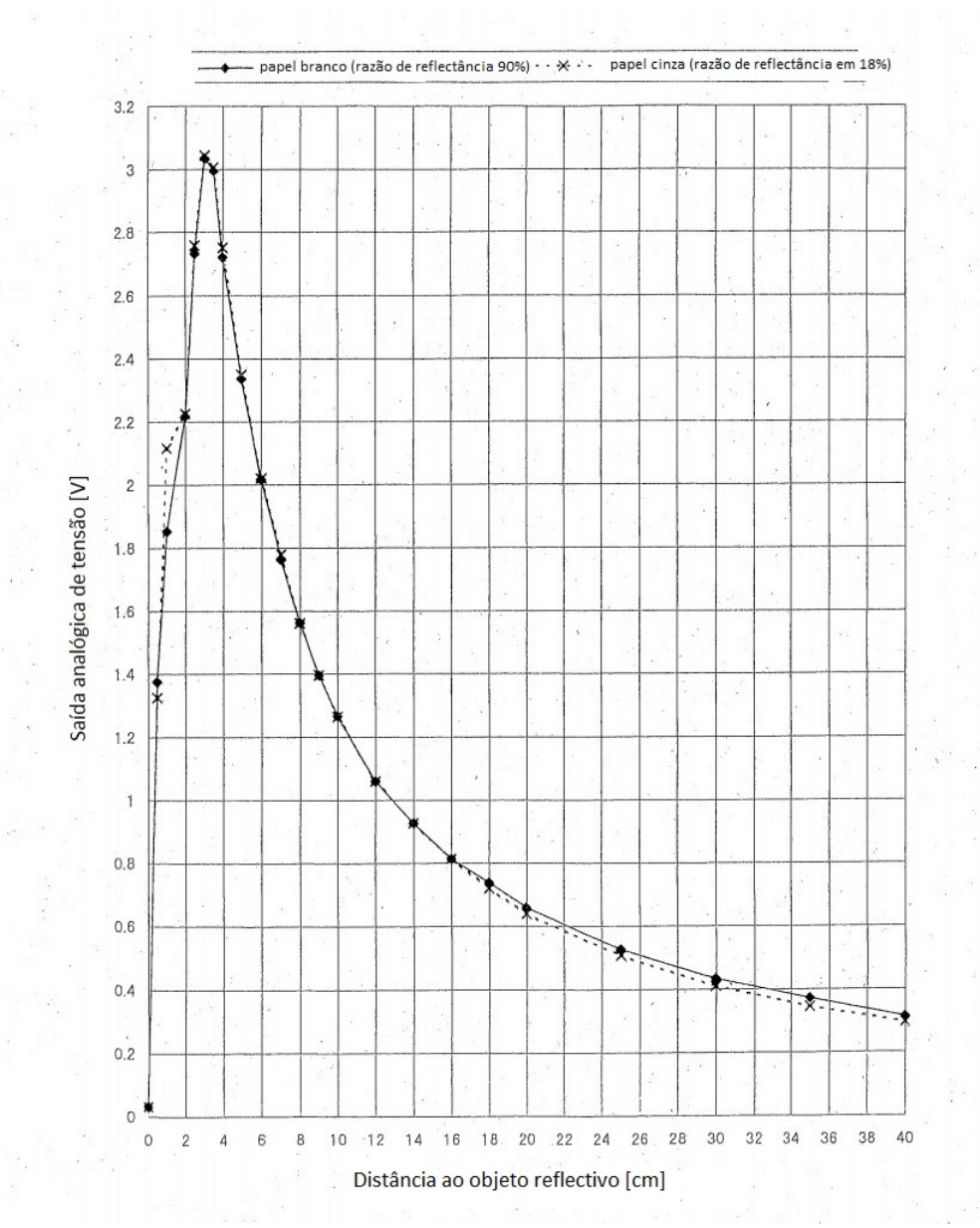

Figura 3.8: Especificação dos sensores infravermelhos (Festo, 2013a)

A equação da curva da figura 3.8 foi estimada, resultando em uma equação polinomial que relaciona distância e tensão, no intervalo (4 cm, 40 cm), (3V, 0.3V), com coeficiente de correlação linear igual a 0.99. A seguir a equação é apresentada:

$$\begin{aligned} D_{est} &= 31.828v^6 - 321.81v^5 + 1325.2v^4 - 2871.8v^3 + 3530.4v^2 - \\ &- 2463.4v + 899.24 \end{aligned} \tag{3.10}$$

onde D_{est} representa a distância estimada do sensor para o obstáculo em milímetros, v^k a tensão do sensor de distância, em volts^k, k a potência de v e as constantes de unidade mm/v^k.

Curva de Bézier

A ordem da curva de Bézier também pode influenciar a forma com que o robô contorna o obstáculo. Uma curva de ordem elevada pode funcionar, mas apresenta um maior número de pontos de controle, maior número de operações matemáticas e portanto um custo computacional maior. Pensando-se em não elevar o custo computacional, além da simplicidade de implementação, optou-se pela curva de Bézier de terceira ordem, que satisfez as necessidades do trabalho. Para uma curva de Bézier de terceira ordem, quatro pontos de controle serão necessários. Para se calcular os pontos de controle $Pc_0 = (A_0, B_0)$, $Pc_1 = (A_1, B_1)$, $Pc_2 = (A_2, B_2)$, $Pc_3 = (A_3, B_3)$ deve-se estimar a distância do Robotino® para o obstáculo, e depois calcular as componentes na direções X e Y.

O cálculo das componentes é mostrado a seguir:

$$D_{estXi} = fa_i D_{est} cos(\pi DS_i/180) \quad (3.11)$$

$$D_{estYi} = fa_i D_{est} sin(\pi DS_i/180) \quad (3.12)$$

onde D_{estXi} representa a distância estimada do sensor i para o obstáculo na direção X, D_{estYi} a distância estimada do sensor i para o obstáculo na direção Y, fa_i um número que é 0 se nenhum obstáculo for detectado ou 1 se algum obstáculo for detectado (adimensional). O índice i representa o respectivo sensor, DS_i o ângulo da direção referente ao sensor i.

As equações dos pontos de controle são definidas a seguir:

$$A_0 = odometry_x \tag{3.13}$$

$$B_0 = odometry_y \tag{3.14}$$

$$A_1 = A_0 + D_{estX0} + fa_0 Rcos(D_{P0}) + D_{estX1} + fa_1 Rcos(D_{P1}) + ... + D_{estX8} + fa_8 Rcos(D_{P8}) \tag{3.15}$$

$$B_1 = B_0 + D_{estY0} + fa_0 Rsin(D_{P0}) + D_{estY1} + fa_1 Rsin(D_{P1}) + ... + D_{estX8} + fa_8 Rsin(D_{P8}) \tag{3.16}$$

$$A_2 = \frac{A_1 + A_3}{2} \tag{3.17}$$

$$B_2 = \frac{B_1 + B_3}{2} \tag{3.18}$$

$$A_3 = obj_x \tag{3.19}$$

$$B_3 = obj_y \tag{3.20}$$

onde A_i, B_i representa as componentes do ponto de controle i, R um parâmetro que regula a distância que o Robotino® contorna o obstáculo, em milímetros, D_{Pi} a direção perpendicular ao sensor i.

De uma forma mais específica, pode-se afirmar que as componentes Pc_0 representam o ponto onde o robô se encontra no momento de detecção do obstáculo. Pc_1 representam um ponto em uma direção perpendicular a indicada pelos sensores. A inclusão da influência de mais de um sensor no cálculo de Pc_1 se dá para o robô se manter em uma trajetória o mais perpendicular possível em relação a face atual do obstáculo. Pc_2 é uma média entre os pontos Pc_1 e Pc_3. Já Pc_3 representa o ponto objetivo, ou ponto de visitação. A figura 3.9 representa de forma ilustrativa a curva calculada.

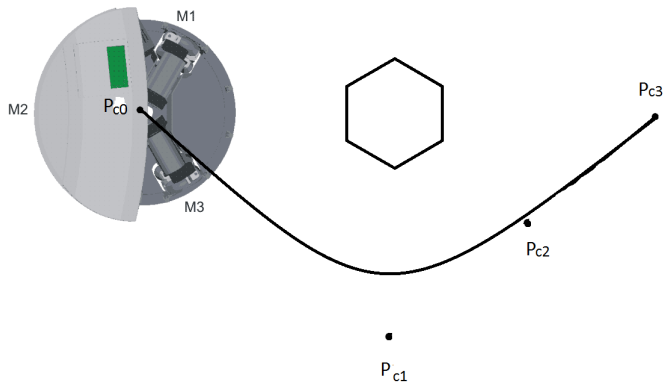

Figura 3.9: Ilustração da curva de Bézier

O lado para o qual o robô desvia do obstáculo é decidido através da análise da direção do robô em relação ao objetivo e o sensor que localizou o obstáculo. Na figura 3.10 é mostrado um exemplo, para o sensor 0, de como a decisão é tomada. Os outros sensores infravermelhos seguem o mesmo princípio.

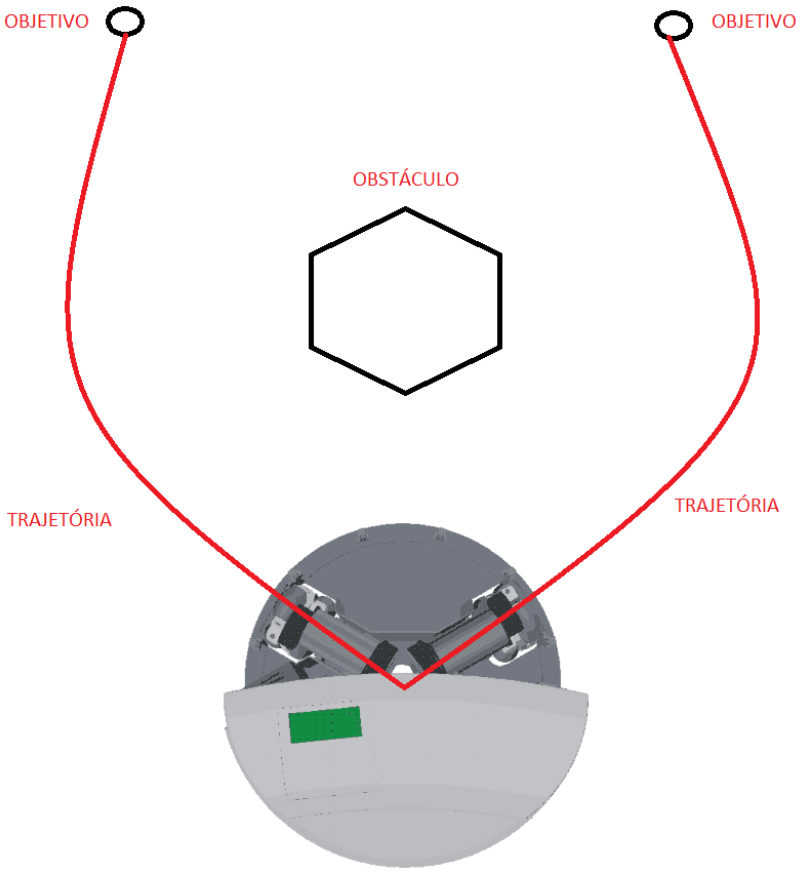

Figura 3.10: Tomada de decisão

Os pontos pelos quais o robô deve percorrer para descrever a curva são então calculados

de acordo com as equações paramétricas em u a seguir.

$$P_x(u) = A_0 + u(-3A_0 + 3A_1) + u^2(3A_0 - 6A_1 + 3A_2) + u^3(-A_0 + 3A_1 - 3A_2 + A_3) \quad (3.21)$$

$$P_y(u) = B_0 + u(-3B_0 + 3B_1) + u^2(3B_0 - 6B_1 + 3B_2) + u^3(-B_0 + 3B_1 - 3B_2 + B_3) \quad (3.22)$$

Quanto menor o incremento do parâmetro u, mais suave a curva será, pois existem mais pontos para descrevê-la, porém isso pode implicar em maior esforço computacional, além de maior imprecisão na detecção da posição pelos sensores do robô.

Vale salientar, que como cada sensor no robô representa uma direção, estima-se a posição do obstáculo, então a curva de Bézier é calculada e assim o robô deve manobrar e desviar do obstáculo. Se por ventura um novo obstáculo for encontrado enquanto o robô descreve a curva, antes de atingir o ponto final desta curva, ela é então recalculada de forma a desviar do novo obstáculo e atingir o ponto final.

3.4 Programa Proposto

O programa foi implementado na linguagem C++, utilizando a API 1.0, na IDE Microsoft Visual Studio 2012, no computador com as configurações apresentadas na tabela 3.2. É composto por 9 funções principais, e, a seguir, é dada uma explicação resumida sobre essas funções. Ao final do tópico será apresentado um fluxograma do programa implementado (figura 3.11), para um melhor entendimento da integração e funcionamento dessas funções.

- `void init()`: Inicializa o Robotino®, realiza a comunicação entre computador via ponto de acesso *wireless lan*, com endereço de IP (*Internet Protocol*) 172.26.201.1. Além disso, a função também é responsável por padronizar a localização atual do robotino em $x = 0$, $y = 0$ e $\phi = 0^o$, ou seja, sempre que inicializado, o robô tomará como origem o lugar em que está, portanto deve-se ter cuidado no momento de ajustar fisicament o robô no momento da inicialização do programa.

- `void CoordReta(float coordx, float coordy, float phi)`: é responsável por mover o Robotino® aproximadamente em linha reta para o próximo objetivo. As entradas `coordx` e `coordx` são calculadas como nas equações 3.2 e 3.3, sendo atualizadas a todo momento, de forma a estar sempre corrigindo possíveis pequenos desvios de rota. As velocidades são calculadas de acordo com as equações 3.6, 3.7, 3.8 e 3.9. Também é responsável por corrigir a orientação do Robotino® regulando a velocidade individual de cada motor, através de uma função da API do Robotino®, garantindo a orientação $-5^o \leq \phi \leq 5^o$ de acordo com o sensoriamento interno do robô (encoder e giroscópio).

- `int CoordParada(float objx, float objy)`: responsável por dizer se o objetivo foi alcançado ou não com a margem aceitável, de acordo com as equações 3.4 e 3.5. Se o objetivo tiver sido alcançado, a função retorna um valor que serve como controle para o próximo passo (parar ou seguir para o próximo ponto de visitação).

- `float estdist(float volts)`: estima a distância do Robotino® para o obstáculo de acordo com as equações 3.11 e 3.12, utilizadas também para o cálculo das curvas de Bézier. Utiliza como entrada o valor de tensão referente ao sensores que detectam o osbtáculo.

- `void DistanceVoltage(float coordx, float coordy, float objx, float objy)`: responsável pela detecção dos obstáculo através da análise do valor da tensão dos sensores e pela geração da rota local de acordo com as equações 3.21 e 3.22. Utiliza como entrada além de `coordx` e `coordy`, os objetivos `objx` e `objy`, que são equivalentes a obj_x e obj_y, explicados nas equações 3.2 e 3.3.

- `void drive()`: faz a ligação entre os algoritmos genéticos a e as funções para a navegação: `void CoordReta(float coordx, float coordy)`, `int CoordParada(float objx, float objy)`, `void DistanceVoltage(float coordx, float coordy, float objx, float objy)`.

- `void destroy()` finaliza a comunicação entre Robotino® e computador depois que o objetivo principal é atingido.

- `int main()` é a função principal do programa, padrão da linguagem de programação escolhida (C++).

A seguir é apresentado o fluxograma do programa implementado (figura 3.11).

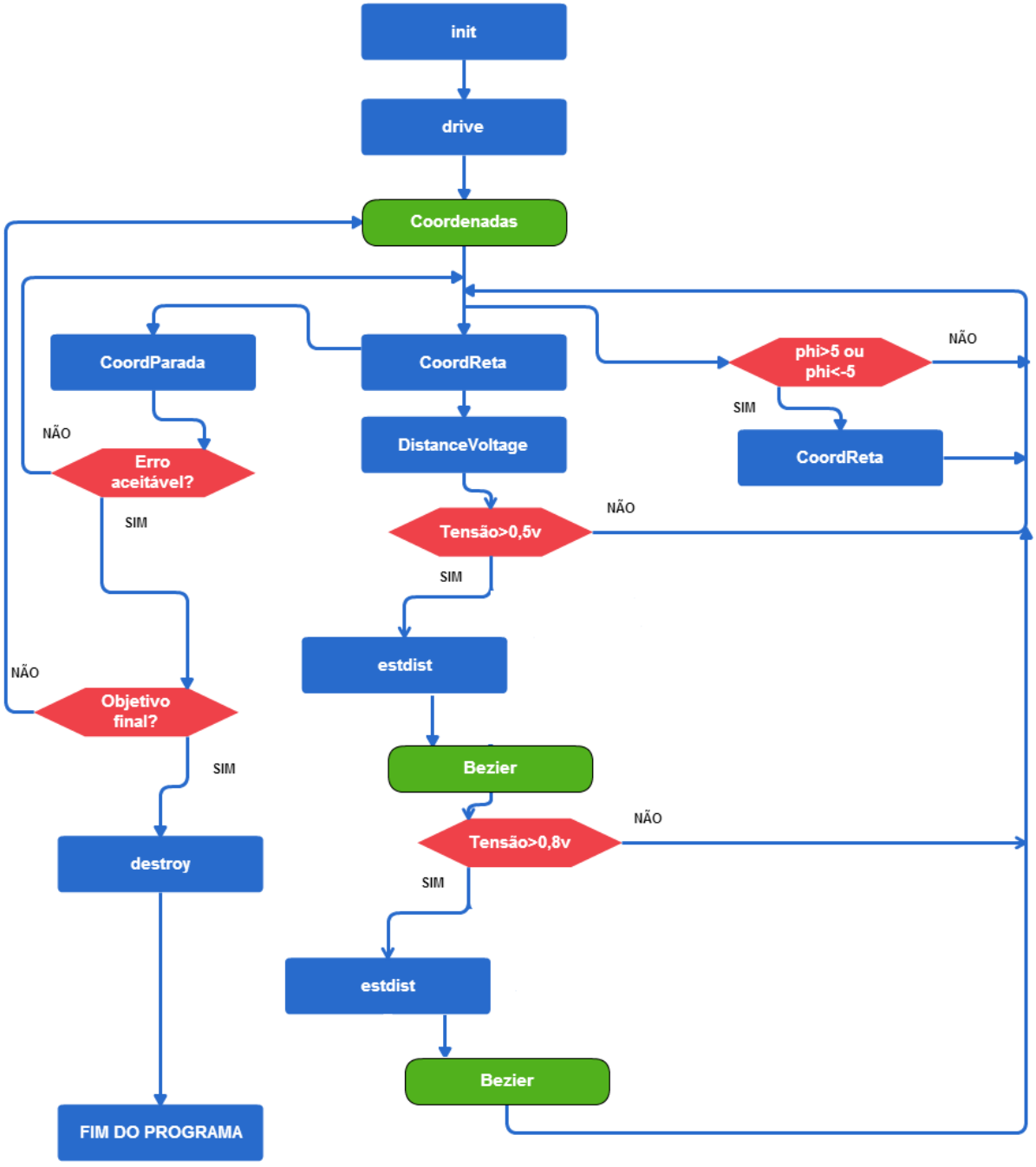

Figura 3.11: Fluxograma

3.5 Metodologia experimental

O algoritmo genético possui vários parâmetros que devem ser ajustados, são eles: população, tamanho do cromossomo, método de seleção, tipo de crossover, taxa de crossover, taxa de mutação, critério de parada e número de iterações. O tamanho de cromossomos está ligado ao número de pontos a ser visitados, assim, fixamos o tamanho do cromossomo em 8, sendo portanto 8 pontos de visitação, com o primeiro sendo o ponto inicial e o último sendo o ponto objetivo. O método de seleção será o de regime permanente conforme proposto por Sivanandam e Deepa (2008). São selecionados dois pais, considerados os melhores indivíduos da geração, ou seja com a maior aptidão atual (menor distância). Depois dá-se início a operação de *crossover*. Como indicado por vários autores (Sivanandam e Deepa, 2008; Von Zuben, 2011; Linden, 2008), o operador de *crossover* selecionado foi o PMX (*partially mapped crossover*), pois esse operador evita que a prole (novas rotas) geradas possam ter problemas como pontos de visitação repetidos (visitados mais de uma vez em uma mesma rota) e não visitados, o que invalidaria a rota. O critério de parada escolhido foi o número máximo de iterações.

Para os parâmetros restantes, foram realizados ajustes com um certo número de opções. O parâmetro de interesse era variado, enquanto os outros permaneciam fixos. No momento em que o valor do parâmetro era ajustado, esse passava a ser o novo padrão e era fixado para definir os outros parâmetros. Nas simulações de ajuste de parâmetros um mesmo conjunto de coordenadas é utilizado. Para fazer comparações entre as simulações do algoritmo genético, define-se a DMI ou Distância Média Inicial como sendo a distância média da primeira iteração e DMF sendo a Distância Média Final como a distância média da iteração final, além de DR como Decréscimo Relativo em relação a população inicial. Assim, temos:

$$DMI = \frac{D_{rota_{1i}} + D_{rota_{2i}} + ... + D_{rota_{popi}}}{pop} \tag{3.23}$$

$$DMF = \frac{D_{rota_{1f}} + D_{rota_{2f}} + ... + D_{rota_{popf}}}{pop} \tag{3.24}$$

$$DR(\%) = |\frac{DMI - DMF}{DMI}|100 \tag{3.25}$$

onde, $D_{rota_{ki}}$ a distância percorrida da rota, calculado como na equação 3.2, para o índividuo k da população inicial, $D_{rota_{kf}}$ distância percorrida pela rota, calculado como na equação 3.2, para o índividuo k da população final, *pop* o tamanho da população.

Nestas equações não é levado em consideração o percurso gerado pelas rotas locais (desvio de obstáculos), já que inicialmente não se tem conhecimento da localização dos obstáculos.

Para a população, foram testados 4 valores distintos: 50, 100, 200, 500. Esses valores não tem nenhuma relação em especial, apenas foram definidos através de observações e experiência do autor. Para cada valor, foram realizadas 100 simulações, e no final uma análise do resultado foi feita com uma conclusão sobre qual valor se adequa melhor ao algoritmo.

Para o número de iterações, a mesma metodologia do parágrafo anterior foi aplicada, com 3 valores distintos: 500, 1000, 3000. Mais uma vez, esses valores não tem nenhuma relação em especial, apenas foram definidos através de observações e experiência do autor.

Para a taxa de mutação o mesmo procedimento foi aplicado, com 4 valores distintos: 0%, 0.1%, 1%, 5%. Esses valores de taxa de mutação são típicamente citados pela literatura (Sivanandam e Deepa, 2008; Von Zuben, 2011; Linden, 2008; Siciliano, 2006).

Para a taxa de crossover, foram testados 4 valores distintos: 25%, 50%, 75%, 100%. Esses valores são típicamente citados pela literatura (Sivanandam e Deepa, 2008; Von Zuben, 2011; Linden, 2008; Siciliano, 2006).

Depois de ajustar o algoritmo genético, foi testado o algoritmo de navegação. Para avaliar a curva de Bézier, ou geração de rotas locais, o robô foi levado a uma rota de colisão e então o seu comportamento para cada parâmetro da curva foi observado. Os parâmetros no algoritmo são, `thresh` (limite de tensão referente a detecção de obstáculos), `dist` (equivalente ao R apresentado nas equações 3.15 e 3.16), `incre2` (número de pontos para a construção da curva). Esses parâmetros foram criados para realizar um ajuste mais fino da curva, e são apresentados detalhadamente no capítulo 4.

Para `thresh`, foram testados 3 valores distintos: 0.3V, 0.6V, 0.9V. Esses números foram escolhidos baseados na figura 3.8 e em experimentações no simulador e no Robotino®. Para cada valor, uma curva será gerada, analisada e depois aplicada ao simulador e ao

Robotino® e então o melhor valor será aplicado no programa. O valor será escolhido através de observação da curva teórica gerada.

Para `dist`, foram testados 4 valores distintos: 300mm, 400mm, 500mm e 600mm. Esses números foram escolhidos baseados em experimentações no simulador e no Robotino®. O mesmo procedimento do parágrafo anterior foi aplicado na avaliação desta variável.

Para `incre2` foram testados: 4, 8, 16, sempre seguindo a mesma metodologia.

Depois de observado o comportamento do algoritmo de navegação, o mesmo é aplicado junto ao algoritmo genético em uma simulação final, em um novo conjunto de pontos. Após 100 simulações, uma avaliação do algoritmo genético e do erro de posição do objetivo final serão apresentadas.

No Robotino®, não há necessidade de reavaliar o ajuste dos parâmentros do algoritmo genético, visto que essa parte funciona de forma *offline*. O conjunto de experimentos finais acontece com a junção do algoritmo genético mais o algoritmo de navegação aplicados no robô real, onde deve visitar um novo conjunto de pontos, com o algoritmo genético decidindo qual rota será percorrida. O percurso será maior e terá mais obstáculos que os testes anteriores. Se houver necessidade, os parâmetros da curva de Bézier podem ser ajustados. Após 20 experimentos, uma avaliação do algoritmo genético e do erro de posição do objetivo final serão apresentadas.

No capítulo seguinte, são apresentados os resultados e a análise da aplicação do programa proposto.

Capítulo 4

Resultados e Discussões

O programa pode ser dividido em dois módulos distintos: **algoritmo genético** e **algoritmo de navegação**. Os algoritmos genéticos geram a ordem em que os pontos de interesse devem ser visitados, e o algoritmo de navegação gerencia a execução da movimentação do robô. Sendo assim, podem ser analisados separadamente, pois a geração da ordem de visitação dos pontos acontece antes de o Robotino® se movimentar, ou seja, quando começa a se movimentar, toda a ordem dos pontos já está traçada. Primeiramente vamos analisar o algoritmo genético e em seguida o algoritmo de navegação.

Para um melhor entendimento das figuras a seguir, a orientação do Robotino® foi desenhada na tela do simulador (figura 4.1), sendo o ponto de partida e de referência a origem desta orientação. O sistema de orientação interna do robô deve estar alinhado com o sistema de referência do ambiente. Um outro sistema de orientação é visto no canto inferior direito da figura 4.1, mas esse diz respeito a posição da câmera do simulador. O ambiente do simulador é mostrado na figura 4.1 a seguir, onde cada quadrado da grade tem lado de tamanho de 0.5 metros de comprimento, e as paredes brancas delimitam o local de trabalho.

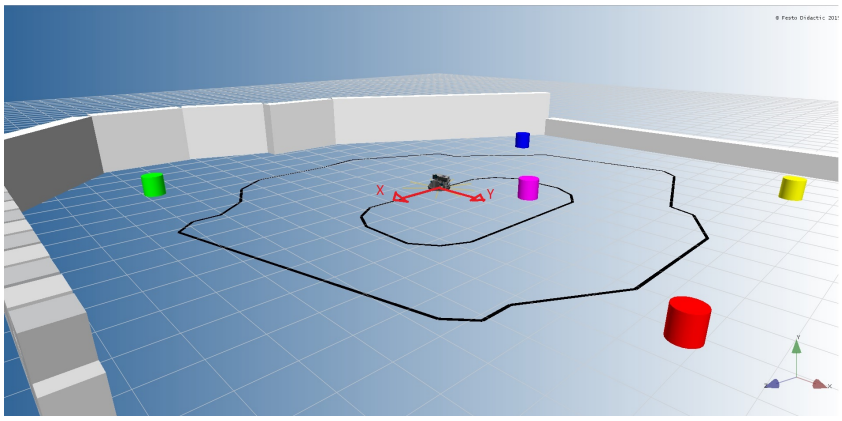

Figura 4.1: Orientação do Robotino®

4.1 Ajuste de Parâmetros do Algoritmo Genético

Todas as simulações foram feitas com os pontos mostrados na tabela 4.1, sendo que o primeiro ponto é o ponto inicial, e o último o final, não podendo ser trocados de lugar (4.2). O algoritmo genético deve organizar a ordem dos pontos de visitação, nos quais a menor distância percorrida será considerada a melhor rota. Para se aproximar da melhor rota, é necessário um ajuste de parâmetros do algoritmo. Um conjunto de 100 testes para cada mudança de parâmetro é realizado e as médias das distâncias finais (DMF) foram apresentadas e comparadas, assim o parâmetro que levar a uma menor média das distâncias finais é escolhido para o ajuste do algoritmo.

Pontos de visitação	X	Y
1	0	0
2	1400	5000
3	-2000	3000
4	-1700	3000
5	4000	-3500
6	4000	-3300
7	-4250	-1400
8	-4000	4500

Tabela 4.1: Pontos de visitação

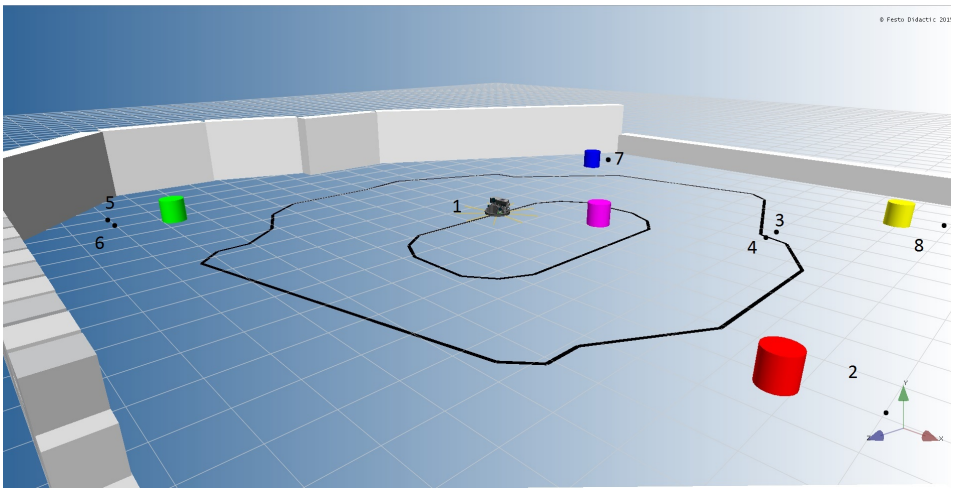

Figura 4.2: Pontos de visitação®

4.1.1 Tamanho da população

Para decidir o tamanho da população foram realizadas 100 simulações para cada número diferente de população (50, 100, 200, 500). Na tabela 4.2 estão definidos os parâmetros utilizados no algoritmo genético. Nas tabelas 4.3, 4.4, 4.5, 4.6, são apresentados os resultados (média, desvio pafrão, moda, frequência), nas quais DMI, DMF e DR foram definidos na seção 3.5.

Parâmetro	Valor
População	50, 100, 200, 500
Tamanho do cromossomo	8
Método de seleção	Regime permanente
Crossover	PMX
Taxa de *crossover*	100%
Taxa de mutação	1%
Critério de parada	Número máximo de iterações
Número de iterações	1000

Tabela 4.2: Parâmetros da simulação (populacão)

Rota	DMI (mm)	DMF (mm)	DR (%)
Média	42899.4	29346.2	31.6
Desvio Padrão	1020.5	1150.7	2.7
Moda	N/A	28374.6	N/A
Frequência	N/A	9	N/A

Tabela 4.3: Resultados população de tamanho 50

Rota	DMI (mm)	DMF (mm)	DR (%)
Média	42895.5	28798.8	32.8
Desvio Padrão	738.2	657.5	1.7
Moda	N/A	28518.6	N/A
Frequência	N/A	14	N/A

Tabela 4.4: Resultados população de tamanho 100

Rota	DMI (mm)	DMF (mm)	DR (%)
Média	43069.6	28545.8	33.7
Desvio Padrão	517.4	313.9	1.1
Moda	N/A	28252.8	N/A
Frequência	N/A	28	N/A

Tabela 4.5: Resultados população de tamanho 200

Rota	DMI (mm)	DMF (mm)	DR (%)
Média	43069.4	28317.6	34.2
Desvio Padrão	297.4	112.9	0.5
Moda	N/A	28252.8	N/A
Frequência	N/A	61	N/A

Tabela 4.6: Resultados população de tamanho 500

Pode-se percerber que quanto maior a população, menor o percurso a ser traçado pelo robô (menor DMF) e mais consistentes são os dados (menor desvio padrão), além de maior repetição (frequência) da melhor resposta possível (28252.8 mm). Isto pode ser explicado pelo fato de que quanto maior a população maior a variabilidade de material genético, ou seja, maior quantidade de informações diferentes, aumentando o espaço de busca. Em uma população pequena o risco de se cair em um ótimo local é maior, como pode ser visto nos casos para população de tamanho 50 e 100, por exemplo. Já com uma população grande a chance de se encontrar rotas iguais ou próximas ao ótimo global aumentam (a partir da população de tamanho 200 a rota que mais se repetiu foi a menor rota possível). Assim o valor do tamanho da população escolhido para as simulações e experimentos é o de 500, onde em 61% dos casos a resposta dada pelo algoritmo foi o ótimo global.

4.1.2 Número de iterações

O mesmo procedimento utilizado na seção anterior foi aplicado para o número de iterações. Os testes foram realizados com a população de 500 e os outros parâmetros

foram mantidos constantes, com o tamanho da população já definido, variando-se apenas o número de iterações (tabela 4.7). Os resultados são apresentados nas tabelas 4.8, 4.9, 4.10.

Parâmetro	Valor
População	500
Tamanho do cromossomo	8
Método de seleção	Regime permanente
Crossover	PMX
Taxa de *crossover*	100%
Taxa de mutação	1%
Critério de parada	Número máximo de iterações
Número de iterações	500, 1000, 3000

Tabela 4.7: Parâmetros da simulação (número de iterações)

Rota	DMI (mm)	DMF (mm)	DR (%)
Média	43009.3	28339.9	34.1
Desvio Padrão	279.7	153.8	0.5
Moda	N/A	28252.8	N/A
Frequência	N/A	50	N/A

Tabela 4.8: Resultados para número de iterações igual a 500

Rota	DMI (mm)	DMF (mm)	DR (%)
Média	43024.8	28326.7	34.2
Desvio Padrão	341.5	126.0	0.6
Moda	N/A	28252.8	N/A
Frequência	N/A	57	N/A

Tabela 4.9: Resultados para número de iterações igual a 1000

Rota	DMI (mm)	DMF (mm)	DR (%)
Média	42996.9	28338.9	34.1
Desvio Padrão	309.5	144.9	0.5
Moda	N/A	28252.8	N/A
Frequência	N/A	51	N/A

Tabela 4.10: Resultados para número de iterações igual a 3000

No caso com 1000 iterações apresentou a menor DMF, o menor desvio padrão e maior repetição do melhor resultado (57 vezes em 100 simulações). No caso com 500 iterações uma explicação para o resultado pior do que no caso com 1000 iterações é a de que não houve iterações suficientes para a convergência em busca de um ótimo global. Nos testes com mais iterações (3000 iterações) não foi observado melhora nos resultados. Isso pode vir

do fato de o algortimo, por exemplo, já ter converjido e no entanto mutações podem ocorrer, causando pertubações e podendo piorar o resultado. O custo computacional também fica maior para um maior número de iterações, porém no valor de 1000 obteve-se um patamar aceitável de confiança (bons resultados) e desempenho (custo computacional).

4.1.3 Taxa de mutação

A seguir, são apresentados os testes para definir a taxa de mutação, que utilizou-se dos valores de: 0%, 0.1%, 1%, 5 %. Os parâmetros para as simulações referentes a taxa de mutação estão expostos na tabela 4.11. Os resultados são apresentados nas tabelas 4.12 4.13, 4.14, 4.15.

Parâmetro	Valor
População	500
Tamanho do cromossomo	8
Método de seleção	Regime permanente
Crossover	PMX
Taxa de *crossover*	100%
Taxa de mutação	0 %, 0.1 %, 1%, 5%
Critério de parada	Número máximo de iterações
Número de iterações	1000

Tabela 4.11: Parâmetros da simulação (taxa de mutação)

Rota	DMI (mm)	DMF (mm)	DR (%)
Média	43022.2	28360.0	34.1
Desvio Padrão	317.4	158.6	0.6
Moda	N/A	28252.8	N/A
Frequência	N/A	47	N/A

Tabela 4.12: Resultados para taxa de mutação igual a 0%

Rota	DMI (mm)	DMF (mm)	DR (%)
Média	43049.6	28327.2	34.2
Desvio Padrão	317.2	125.9	0.5
Moda	N/A	28252.8	N/A
Frequência	N/A	54	N/A

Tabela 4.13: Resultados para taxa de mutação igual a 0.1%

Rota	DMI (mm)	DMF (mm)	DR (%)
Média	43034.1	28318.0	34.2
Desvio Padrão	321.4	117.7	0.5
Moda	N/A	28252.8	N/A
Frequência	N/A	58	N/A

Tabela 4.14: Resultados para taxa de mutação de 1%

Rota	DMI (mm)	DMF (mm)	DR (%)
Média	43022.8	28336.6	34.1
Desvio Padrão	301.9	144.8	0.5
Moda	N/A	28252.8	N/A
Frequência	N/A	57	N/A

Tabela 4.15: Resultados para taxa de mutação de 5%

A mutação tem a função de criar perturbações de modo a impedir a convergência prematura para um ótimo local. Se perturbação for muito pequena, como no caso com a taxa de ocorrência de 0 % e 0.1%, não será suficiente para aumentar o espaço de busca e sair do ótimo local. Se a perturbação for muito alta, como no caso com a taxa de ocorrência de 5%, o programa terá dificuldades de convergir. Assim o valor que apresentou o melhor compromisso entre perturbação e convergência foi o de valor de 1%, pois apresentou o menor DMF assim como o menor desvio padrão para o DMF, além do maior número de repetições do ótimo global. É perceptível a melhora dos resultados, mesmo que em pequena escala, quando há a inclusaão da taxa de mutação.

4.1.4 Taxa de *crossover*

O último parâmetro a ser ajustado foi a taxa de *crossover*. Os parâmetros utilizados no algoritmo genético estão expostos na tabela 4.16 e os valores testados foram: 25 %, 50 %, 75%, 100%. Após a apresentação (tabelas 4.17, 4.18, 4.19, 4.20) e a análise dos resultados para a taxa de *crossover* a tabela com todos os parâmetros ajustados é apresentada (tabela 4.21).

Parâmetro	Valor
População	500
Tamanho do cromossomo	8
Método de seleção	Regime permanente
Crossover	PMX
Taxa de *crossover*	25%, 50%, 75%, 100%
Taxa de mutação	1%
Critério de parada	Número máximo de iterações
Número de iterações	1000

Tabela 4.16: Parâmetros da simulação (taxa de *crossover*)

Rota	DMI (mm)	DMF (mm)	DR (%)
Média	43051.4	28307.9	34.2
Desvio Padrão	280.8	96.4	0.5
Moda	N/A	28252.8	N/A
Frequência	N/A	60	N/A

Tabela 4.17: Resultados para taxa de *crossover* de 25%

Rota	DMI (mm)	DMF (mm)	DR (%)
Média	43029.8	28333.8	34.2
Desvio Padrão	288.9	125.2	0.5
Moda	N/A	28252.8	N/A
Frequência	N/A	52	N/A

Tabela 4.18: Resultados para taxa de *crossover* de 50%

Rota	DMI (mm)	DMF (mm)	DR (%)
Média	43019.7	28313.0	34.2
Desvio Padrão	296.7	107.7	0.5
Moda	N/A	28252.8	N/A
Frequência	N/A	62	N/A

Tabela 4.19: Resultados para taxa de *crossover* de 75%

Rota	DMI (mm)	DMF (mm)	DR (%)
Média	43038.2	28321.2	34.2
Desvio Padrão	350.3	134.0	0.6
Moda	N/A	28252.8	N/A
Frequência	N/A	65	N/A

Tabela 4.20: Resultados para taxa de *crossover* de 100%

A partir dos resultados apresentados, pode-se verificar que a simulação com taxa de *crossover* de 25% apresentou melhores resultados (menor DMF, baixo desvio padrão de

DMF e em 60% dos casos apresentou o ótimo global). Já a configuração com 50% de taxa de crossover apresentou os piores resultados (maior DMF e menor número de ótimos globais). Uma taxa baixa demais pode significar que não está havendo troca de informações suficiente entre os indivíduos, ou seja, o espaço de busca não foi bem explorado. Já uma taxa alta demais pode significar que está havendo troca de informações de forma demasiada, ou seja, mesmo o ótimo global sendo encontrado, o algoritmo continua procurando, saindo do ótimo, e pode acabar preso em um mínimo local, podendo haver perda de informações. Assim o valor que, dentre os testados, representou o melhor compromisso entre boa procura no espaço de buscas e manutenção das melhores informações foi o valor de 25%.

Todos os parâmetros ajustados se tornaram padrão para as próximas simulações e experimentos com algoritmo genético. Os valores são mostrados na tabela 4.21.

Parâmetro	Valor
População	500
Tamanho do cromossomo	8
Método de seleção	Regime permanente
Crossover	PMX
Taxa de *crossover*	25%
Taxa de mutação	1%
Critério de parada	Número máximo de iterações
Número de iterações	1000

Tabela 4.21: Parâmetros ajustados do algoritmo genético

4.2 Algoritmo de navegação

O algoritmo de navegação é responsável por gerenciar os movimentos do Robotino®, executando a ordem de visitação definida pelo algoritmo genético, ou seja, a rota global, e criando rotas locais para desvio de obstáculos. A velocidade foi estipulada de modo a manter a segurança do robô e dos usuários, e ficou em 200 mm/s. Foi-se observado no simulador que o Robotino® teria capacidade de andar mais rápido, porém, em testes com o robô físico os impactos em velocidades mais elevadas, mesmo com os sensores do bumper ativados, poderiam danificar o equipamento, sem falar que em caso de travamento do programa o robô deveria ser parado manualmente. Assim, a velocidade de 200 mm/s será utilizada tanto no simulador quanto no robô.

4.2.1 Ajuste dos Parâmetros da curva de Bézier

Antes de apresentar os resultados da navegação de percursos com obstáculos, será feita uma análise sobre o desvio de obstáculos. Para formar a curva de Bézier referente ao obstáculo a ser desviado, alguns fatores podem ser analisados para regular o comportamento da curva. São eles: limites de tensão (chamado no programa de `thresh`, `thresh2` e `thresh3`), distância (`dist` ou R apresentado nas equações 3.15 e 3.16) e número de pontos da curva (`incre2`).

Limite de tensão (detecção de obstáculos)

Os sensores infravermelhos trabalham de acordo com a figura 3.8. Então, percebe-se que a distância do robô para o obstáculo diminui a medida que o valor de tensão aumenta. Se um valor baixo de tensão for escolhido para o limite de tensão, significa que o robô ficará mais responsivo, identificando obstáculos a uma distância mais longa, porém pode acabar identificando obstáculos que não atrapalhariam sua trajetória, estando a uma distância segura. Já com um valor alto para o limite de tensão, o robô ficará menos responsivo, filtrando mais obtáculos, entretanto o robô teria menos espaço hábil para lidar com o obstáculo. Os testes seguintes foram realizados com os parâmetros apresentados na tabela 4.22, para o limite de detecção de obstáculos (`thresh`). A figura 4.3 representa de forma ilustrativa o efeito da variação de tensão.

Variável	Valor	Grandeza	Controla na curva de Bézier
`thresh`	0.3/0.6/0.9	Tensão (V)	A_0, B_0
`thresh2`	0.9	Tensão (V)	A_0, B_0
`thresh3`	1.4	Tensão (V)	A_0, B_0
`dist`	600	Milímetros (mm)	A_1, B_1 e A_2, B_2
`incre2`	16	Adimensional	Suavidade

Tabela 4.22: Parâmetros da curva de Bézier

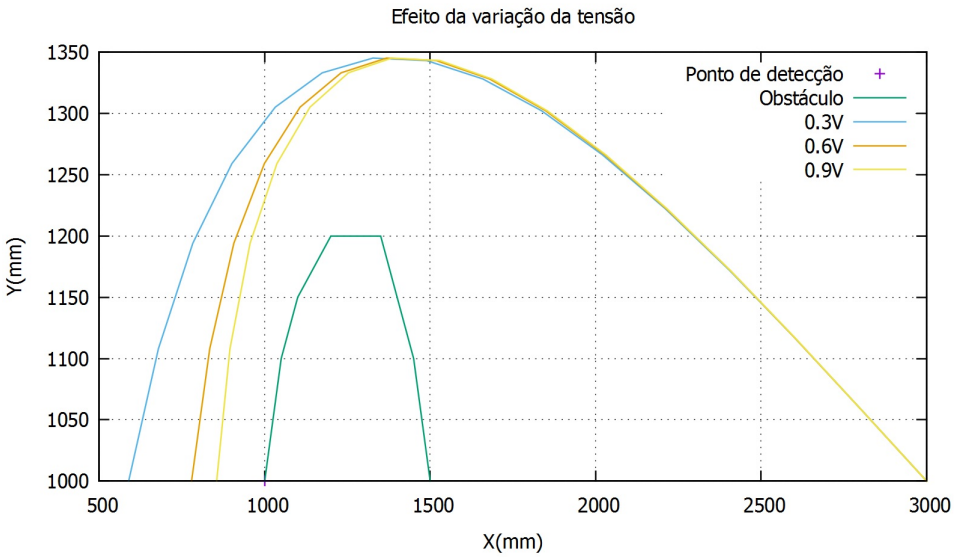

Figura 4.3: Efeito na variação de tensão

Inicialmente, optou-se por trabalhar com o valor mais baixo para o limite de tensão (detecção de obstáculos), no intuito de torná-lo mais responsivo aos obstáculos. No entanto, durante os experimentos realizados com o Robotino®, verificou-se que para este valor o robô acabava identificando obstáculos inexistentes (fantasmas). Com isso, fixou-se o limite de detecção de obstáculos em 0.6 Volt, baseado nos desenhos e experimentos. O limite para recálculo da trajetória (`thresh2`) em 0.9 Volt e o limite de colisão (`thresh3`) em 1.4 Volt, que foram escolhidos com base na figura 3.8. O limite de colisão é um valor que indica que o robô está muito próximo ao obstáculo, e assim que é atingido o robô se distancia numa direção paralela a indicada pelo sensor que detectou o obstáculo, no sentido contrário ao indicado pelo sensor.

Distância

O parâmetro `dist` regula diretamente a distância que o ponto de controle A_1, B_1 fica do obstáculo, funcionando como uma espécie de raio de curvatura. O ponto de controle A_2, B_2 também é mudado de posição, já que em seu cálculo o ponto A_1, B_1 é utilizado. Quanto menor esse valor mais próximo o Robotino® passará do obstáculo forçando o robô a estar constantemente recalculando sua trajetória, praticamente seguindo o formato do obstáculo e assim aumentando o custo computacional do processo. Já para um valor mais alto, mais longe ele passará do obstáculo, podendo ocasionar um desvio muito grande da trajetória original. Se outros obstáculos forem encontrados no processo, o robô pode

acabar não conseguindo encontrar o seu objetivo, pois pode ficar preso em um ciclo infinito. Visto essas possibilidades, não se pode afirmar qual configuração apresentaria o melhor custo computacional. Os testes realizados para a análise das curvas é feito a seguir, onde os valores de 300 mm, 400 mm, 500 mm e 600 mm foram utilizados em `dist`. Na tabela 4.23, são apresentados os parâmetros da curva de Bézier utilizada. Na figura 4.4 o efeito da variação de `dist` é representado de forma mais ilustrativa.

Variável	Valor	Grandeza	Controla na curva de Bézier
`thresh`	0.6	Tensão (V)	A_0, B_0
`thresh2`	0.9	Tensão (V)	A_0, B_0
`thresh3`	1.4	Tensão (V)	A_0, B_0
`dist`	300/400/500/600	Milímetros (mm)	A_1, B_1 e A_2, B_2
`incre2`	16	Adimensional	Suavidade

Tabela 4.23: Parâmetros da curva de Bézier

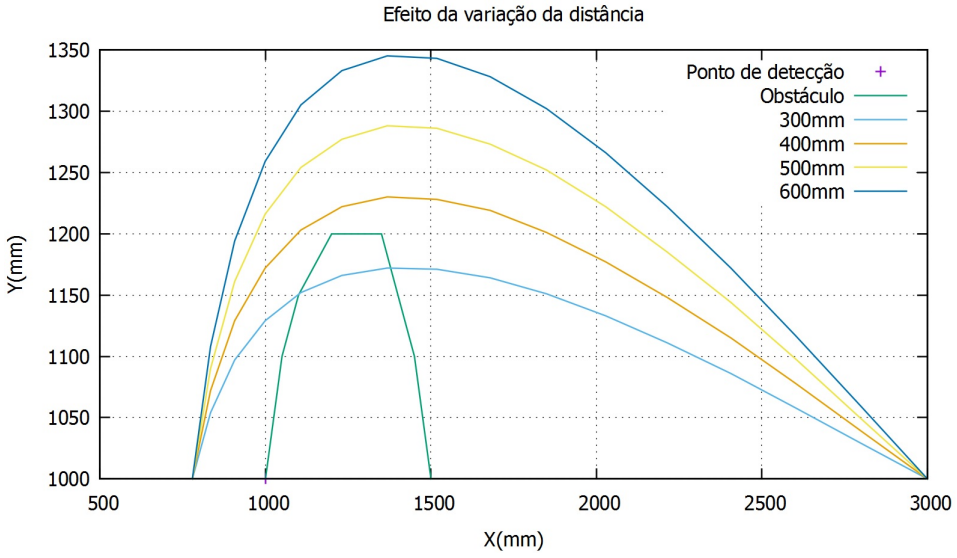

Figura 4.4: Efeito da variação da distância

O valor de 300 mm foi usado com sucesso no simulador, mas no robô real não funcionou muito bem, havendo inclusive colisões, devido a maior influência dos pontos cegos, de fatores externos (iluminação, sujeira nas rodas, imperfeições no piso etc) . Assim, funcionou melhor em ambas as situações (simulação e experimentos) foi o de 400 mm (valor próximo do diâmetro do robô, que é de 370 mm), pois não desviou tanto da trajetória como o valor de 600 mm e nem apresentou colisões como o valor de 300 mm.

Número de pontos da curva

Para descrever uma curva de Bézier de terceira ordem de forma perfeita seriam necessários infinitos pontos. Assiml, fica claro que quanto mais pontos, mais próximo da curva de Bézier e portanto mais suave a curva, pois ela é aproximada por segmentos de reta. Entretanto, isso significa um maior custo computacional, além de gerar imprecisão na posição do robô, pois o mesmo apresentou comportamento instável, tanto nas simulações quanto nos experimentos, girando em torno do ponto sem conseguir atingí-lo. Isso ocorre pois sendo o intervalo muito pequeno poderiam haver vários pontos dentro do erro admissível. Com menos pontos, a curva fica menos suave, entretanto fica mais fácil de ser calculada, diminuindo o custo computacional. Um valor determinado de forma empírica foi pensando de forma a manter o compromisso entre a suavidade da curva e custo computacional. Esse valor foi de 16 pontos formando a curva, assim 15 segmentos de reta constroem a curva. A variável no programa que recebe este valor é a `incre2`. Outra forma de se implementar a curva de Bézier seria fazer com que o robô constantemente variasse sua velocidade de maneira a descrevê-la, o que poderia trazer a vantagem de tornar a curva mais suave e fiel a curva idealizada, porém este método é mais complexo matematicamente, além de ser mais custoso computacionalmente devido a maior quantidade de operações. A seguir são apresentadas a tabela 4.24 com os parâmetros utilizados e a figura 4.5 com a ilustração da diferença do número de pontos.

Variável	Valor	Grandeza	Controla na curva de Bézier
`thresh`	0.6	Tensão (V)	A_0, B_0
`thresh2`	0.9	Tensão (V)	A_0, B_0
`thresh3`	1.4	Tensão (V)	A_0, B_0
`dist`	400	Milímetros (mm)	A_1, B_1 e A_2, B_2
`incre2`	4/8/16	Adimensional	Suavidade

Tabela 4.24: Parâmetros da curva de Bézier

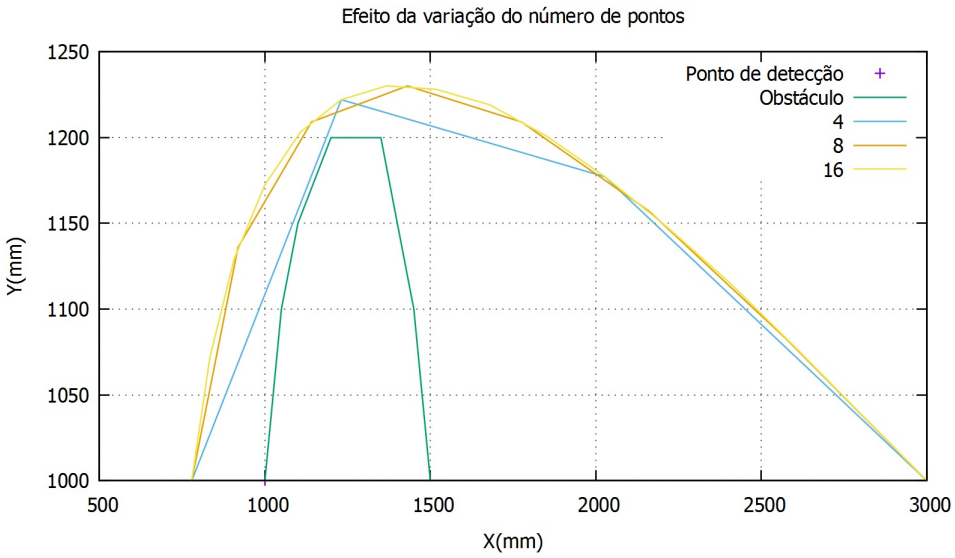

Figura 4.5: Efeito da variação do número de pontos que formam a curva

A tabela 4.25 apresenta os parâmetros ajustados da curva de Bézier.

Variável	Valor	Grandeza	Controla na curva de Bézier
`thresh`	0.6	Tensão (V)	A_0, B_0
`thresh2`	0.9	Tensão (V)	A_0, B_0
`thresh3`	1.4	Tensão (V)	A_0, B_0
`dist`	400	Milímetros (mm)	A_1, B_1 e A_2, B_2
`incre2`	16	Adimensional	Suavidade

Tabela 4.25: Parâmetros da curva de Bézier

4.3 Teste no simulador: algoritmo genético mais algoritmo de navegação

Nesta simulação, o Robotino® saiu do ponto inicial (0,0) e atinge o ponto objetivo (4000, -3500) (tabela 4.26), ou seja, não se trata de um circuito. Foram verificados a performace do algoritmo genético e o erro de posição do robô em relação ao objetivo final. Esta simulação ocorreu com a integração entre o algoritmo genético e o algoritmo de navegação, onde tivemos 100 testes realizados, de onde os dados foram extraídos. Um conjunto de coordenadas é dado e então algoritmo genético as organiza de modod a achar uma boa rota, e então passa a ordem de visitação para o algoritmo de navegação, que realiza a execução do percurso. Na tabela 4.27 são apresentadas a média, desvio padrão, moda e frequência de DMI, DMF e DR, e na tabela 4.28 são apresentados média e

desvio padrão de, ES_X (Erro em módulo da posição retornada pelo odometria em relação ao objetivo no eixo X), ES_Y (Erro em módulo da posição retornada pela odometria em relação ao objetivo no eixo Y). Na figura 4.6 é ilustrada a melhor rota retornada pelo algoritmo genético.

Pontos de visitação	X	Y
1	0	0
2	1400	5000
3	-4000	4500
4	-1700	3000
5	2550	3000
6	4000	-3300
7	-2000	1000
8	4000	-3500

Tabela 4.26: Pontos de visitação da simulação

Rota	DMI (mm)	DMF (mm)	DR (%)
Média	39365,1	22132,6	43.8
Desvio Padrão	243.2	1154.1	2.9
Moda	N/A	21399.0	N/A
Frequência	N/A	54	N/A

Tabela 4.27: Resultados do algoritmo genético para simulação

Simulação	Objetivo X (mm)	ES_X (mm)	Objetivo Y (mm)	ES_Y (mm)
Média	4000	3.6	-3500	12.5
Des Pad	N/A	1.8	N/A	4.2

Tabela 4.28: Resultados para percurso com obstáculos, de objetivo (4000, -3500)

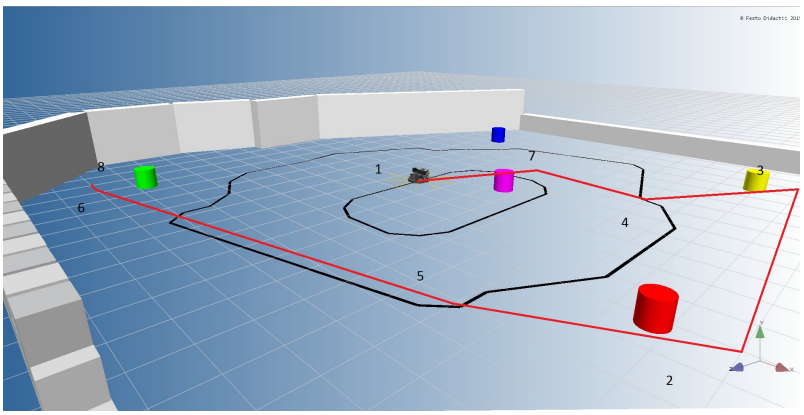

Figura 4.6: Visualização da melhor rota gerada pelo algoritmo genético

A melhor rota foi de 21399.0 mm, e se repetiu 54 vezes dentro das 100 simulações. Considera-se 21399.0 mm como ótimo global, pois foi o menor resultado obtido para essa

configuração pelo algoritmo genético, além de ter sido o resultado que mais se repetiu (moda), demonstrando que o algoritmo genético ajustado, mesmo sem garantias de se encontrar o ótimo global, consegue fazê-lo em mais da metade das simulações. O desvio padrão do DMF foi alto, porém isso é explicado pelas distâncias entre os pontos, onde a troca da ordem de um ponto pode significar maiores distâncias percorridas.

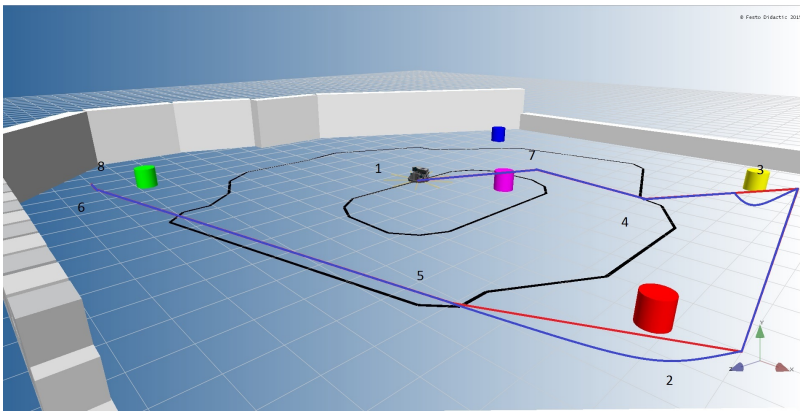

Figura 4.7: Caminho realizado pelo robô (azul)

Na figura 4.7 o Robotino® encontra obstáculos entre o segmento de reta formado entre 4 e 3 e entre 2 e 5. O robô foi capaz de desviar dos obstáculos com êxito e assim seguir rumo ao objetivo. Mesmo com uma rota com 8 pontos de visitação e obstáculos no caminho o algoritmo se manteve dentre o limite de erro, que é a condição de parada para o programa, onde o erro no eixo X e no eixo Y deve ser menor que 20 mm concomitantemente. O robô manteve bem a velocidade estipulada, não sendo necessário frear o robô para desviar dos obstáculos.

4.4 Experimentos com o Robotino®

Conjunto de testes foi realizado no setor IV, Bloco I, sala 6 e 7 da UFRN (figura 4.8), que apresentava um espaço maior, poucas imperfeições no piso e um conjunto de linhas em grade que serviram como parâmetro para definir as coordenadas que o robô deveria visitar. Essas coordenadas estão representadas na figuras pelos números, de acordo com a tabela 4.29).

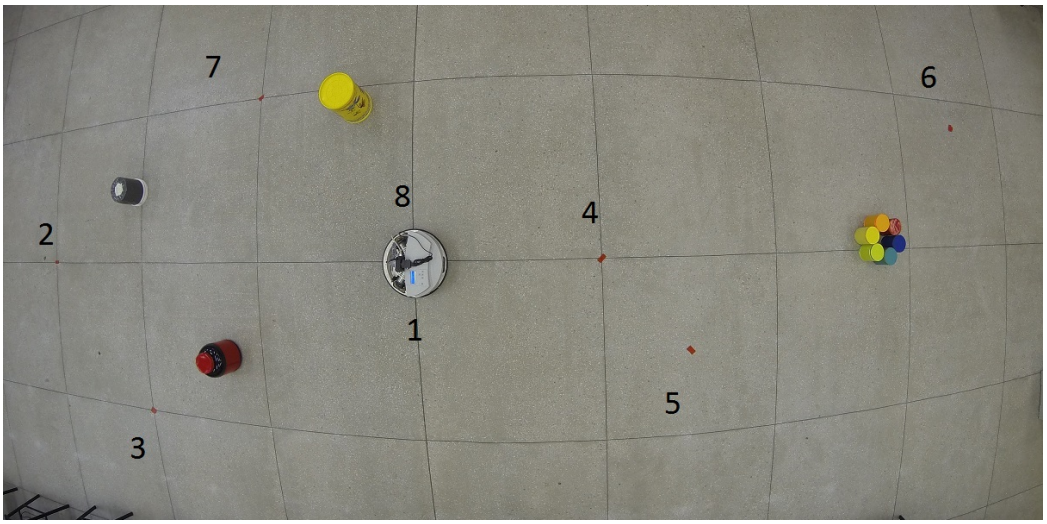

Figura 4.8: Ambiente do experimento

Pontos	X	Y
1	0	0
2	3000	0
3	2000	1000
4	-1000	0
5	-1500	500
6	-3500	-800
7	1000	-1000
8	0	0

Tabela 4.29: Pontos de visitação para experimentos finais com o Robotino®

4.4.1 Experimento com Robotino®: algoritmo genético mais algoritmo de navegação

Para a realização dos experimentos, a rota global foi determinada pelo algoritmo genético (*offline*), levando em consideração os pontos de visitação (tabela 4.29). As rotas locais são descritas segundo curvas de Bézier, de acordo com as informações adquiridas através dos sensores infravermelhos. O robô deve sair da origem, passar por todos os pontos e retornar a origem, realizando, portanto, um circuito. Foram realizados 20 experimentos com o robô real, com os resultados apresentados na tabela 4.30.

Os parâmetros utilizados pelo algoritmo genético nos experimentos com o robô são os mesmo utilizados nas simulações (tabela 4.21). Conforme explicado anteriormente, o algoritmo genético roda de forma *offline* e calcula a rota com base nas coordenadas fornecidas, não levando em consideração os dados fornecidos pelo robô. Sendo assim,

nenhum ajuste foi aplicado aos experimentos com o robô real em relação aos parâmetros que forami apresentados nas simulações.

Para a curva de bézier os valores utilizados para os experimentos com o robô real foram os mesmo ajustados para o simulador e estão listados na tabela 4.25.

A seguir são apresentados os resultados do algoritmo genético na tabela 4.30 e dos valores referentes ao objetivo (posição e erros) nas tabelas 4.31, 4.32, nas quais ER_X (mm) e ER_Y (mm) são os erros em módulo em relação às coordenadas desejadas nos eixos X e Y, respectivamente, através de marcações da posição do robô, em milímetros; ERS_X (mm) é o módulo de ES_X (mm) menos ER_X (mm); ERS_Y (mm) é o módulo de ES_Y (mm) menos ER_Y (mm). ERS_X, ERS_Y servem para mostrar a diferença entre os resultados obtidos pea odometria e os medidos através das marcações.

A medição da posição do robô se dava a partir de marcações em dois pontos do robô, assim era possível descobrir a localização do seu centro e comparar com a marca feita no piso.

Rota	DMI (mm)	DMF (mm)	DR (%)
Média	20487.9	14520.1	29.1
Desvio Padrão	108.1	69.0	0.5
Moda	N/A	14483.1	N/A
Frequência	N/A	15	N/A

Tabela 4.30: Resultados do algoritmo genético para experimento com o Robotino®

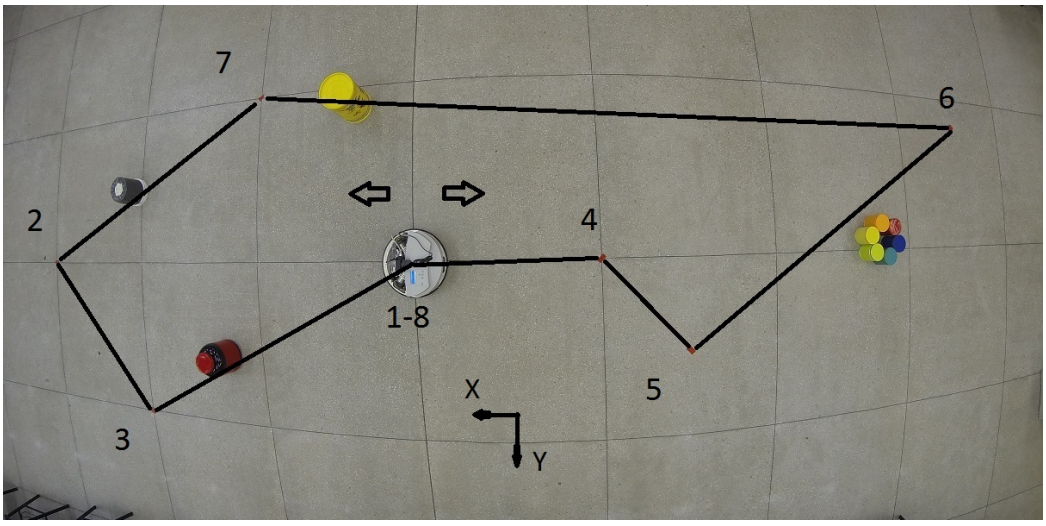

Figura 4.9: Melhor rota para experimento com Robotino®

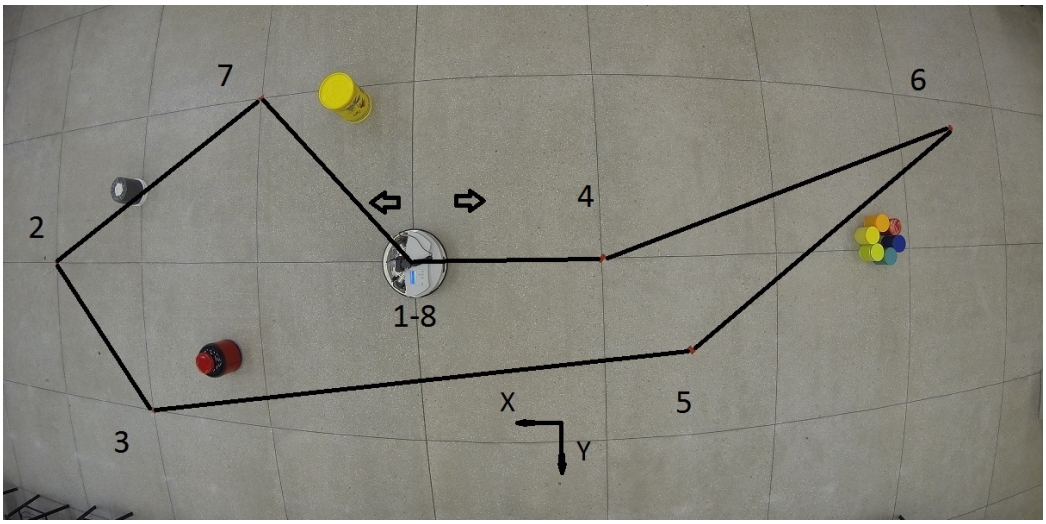

Figura 4.10: Outra rota sugerida pelo algoritmo genético

Nesta configuração o DR (Decréscimo Relativo) ficou na casa de 29.1%, mostrando a melhora das rotas geradas em relação as rotas aleatórias. A melhor rota apresentada foi a de 14483.1 mm de distância, que se repetiu 15 vezes, e é ilustrada na figura 4.9. Outra rota apresentada foi a de 14610.2 mm de distância, que se repetiu 4 vezes (figura 4.10). A outra rota apresentou valor de 14713,7 de distância percorrida. Não foi implementado uma limitação do sentido que o robô deveria seguir, existindo portanto duas rotas possíveis para cada valor de distância percorrida. O desvio padrão do DMF foi baixo, devido ao grande número de repetições da melhor rota, aliado as outras rotas que possuem valores DMF próximos entre si. Para esta configuração a rota com melhor resultado se repetiu 15 vezes, confirmando o que foi discutido na seção 4.1.5, de que a moda do DMF representa a melhor resposta possível.

A seguir os resultados referentes ao ponto de chegada (0,0), para o algoritmo de navegação (tabelas 4.31 e 4.32).

Exp	Objetivo X (mm)	ES_X (mm)	ER_X (mm)	ERS_X (mm)
Média	0.00	16.87	23.59	24.81
Des Pad	N/A	2.40	18.69	21.61

Tabela 4.31: Resultado dos experimentos para o eixo X

Exp	Objetivo Y (mm)	ES_Y (mm)	ER_Y (mm)	ERS_Y (mm)
Média	0.00	7.54	37.06	37.17
Des Pad	N/A	5.03	34.42	35.25

Tabela 4.32: Resultados dos experimentos para o eixo Y

Os valores retornados pela odometria se enquadraram dentro do valor de erro admissível para os mesmos. Conforme mencionado anteriormente, essa condição foi implementada diretamente no programa e faz parte da condição de parada, na qual a tolerância é de 20 mm para a direção X e Y. O erro em média apresentado pela odometria foi maior na direção X e os valores de desvio padrão na direção X e Y foram pequenos. Os erros reais em X e em Y também foram maiores, sendo que a direção Y apresentou o maior valor em média. Isso pode ter ocorrido devido a maior dificuldade de alinhar o robô nessa direção no momento da partida, incorrendo em mais erros no eixo Y. Para as medições realizadas através das marcações no piso foi considerado como tolerância admissível o valor de 40mm para X e para Y, pois é um valor que corresponde a aproximadamente 10% do diâmetro do robô. Assim, em nenhum momento os valores de erro real no eixo X apresentaram valores maiores que a tolerância admissível, porém no eixo Y em vários momentos esses valores superaram os admissíveis, mesmo assim, em média, ambos estão dentro da margem estipulada. A diferença entre os valores retornados pela odometria e os medidos através das marcações também se enquadraram dentro do admissível (20% do diâmetro do robô) em média, mesmo com grandes diferenças em alguns testes.

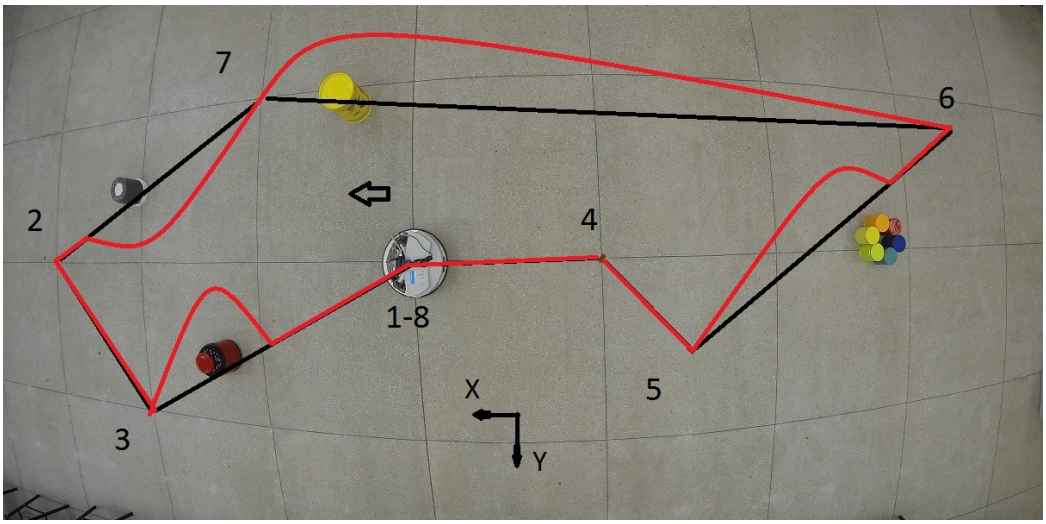

Figura 4.11: Rota aproximadamente executada pelo Robotino® (vermelho)

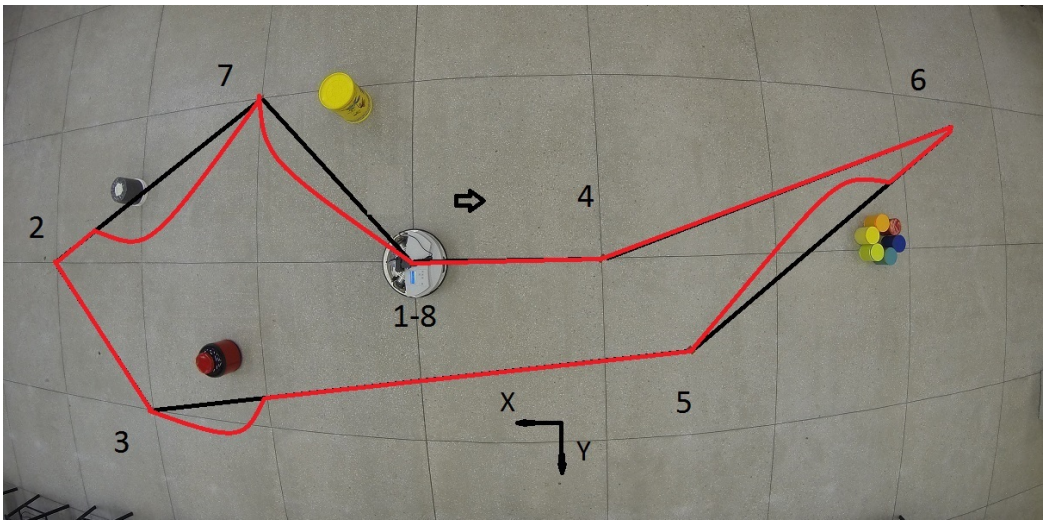

Figura 4.12: Outra rota aproximadamente executada pelo Robotino® (vermelho)

Nas rotas apresentadas (figuras 4.11, 4.12) o robô está em rota de colisão com 4 obstáculos, sendo necessários identificá-los ao longo do percurso. Os valores da tabela 4.25 de `thresh`, `thresh2` e `thresh3` se mostraram adequados para a identificação dos obstáculos. Quando apenas o primeiro limite era atingido (`thresh`), a rota local era um pouco mais longa, porém mais suave e de custo computacional mais baixo, pois não era necessário recalcular a curva. Isso ocorria quando o obstáculo ficava alinhado com algum dos sensores (direção do sensor alinhada com a normal da superfície). Quando o segundo limite era atingido (`thresh2`), a rota de desvio se assemelhava ao formato do obstáculo e era mais curta em relação a curva do primeiro limite (`thresh`), porém menos suave e de custo computacional mais alto. Quando o terceiro limite(`thresh3`) era atingido, o robô estava muito próximo de colidir com o obstáculo, e então o robô se afastava para depois recalcular a curva. O valor de `dist` proporcionou um desvio mais rente ao obstáculo, assim a rota ficava sendo recalculada, o que proporcionava maior segurança, mas aumentava o custo computacional. O valor do número de pontos proporcionou uma construção da curva suave e factível para o robô. O robô passou mais perto dos obstáculos e ficou mais perto da rota original, porém ocursto computacional foi maior e a suavidade da curva foi reduzida. Assim, o algoritmo de navegação conseguiu cumprir a rota global estipulada pelo algoritmo genético com sucesso, realizando os devidos ajustes com as curvas de Bézier para desviar dos obstáculos.

Capítulo 5

Conclusões

A utilização dos algoritmos genéticos para a otimização das rotas se mostrou como uma poderosa ferramenta. Os algoritmos genéticos possuem um bom compromisso entre direcionar a pesquisa das respostas para espaços de busca interessantes do ponto de vista da aptidão, e vasculhar todo o espaço de busca de resposta. Isso faz com que esses algoritmos se tornem versáteis e possam ser utilizados para resolver problemas em que a descrição matemática seja impossível, ou de custo muito alto, uma vez que os algoritmos genéticos se baseiam no que se quer ver na resposta e não na descrição do problema em si. Mesmo assim, não há nenhuma garantia de que os algoritmos genéticos funcionem igualmente sempre, como pode ser visto nas tabelas de resultados, onde nem sempre o resultado foi igual. O algoritmo genético é altamente dependente dos parâmetros impostos pelo programador, como foi visto nos casos de número de iterações, taxa de *crossover*, taxa de mutação e tamanho da população, onde a variação destes parâmetros ocasionava mudanças nas respostas. Outro fato a ser considerado é que também não há garantia de que o ótimo global seja encontrado, apenas pode-se aumentar a chance de se encontrá-lo.

A otimização da rota tomada implica redução do caminho percorrido, que pode trazer vantagens como economia de combustível, redução do desgaste de peças, e portanto economia de dinheiro e equipamento. Logicamente, no mundo real, vários outros fatores devem ser analisados, como por exemplo a prioridade de execução e o tempo. Tais fatores deveriam ser levado em consideração no momento de implementação da função objetivo, dependendo do que for requerido. Para os fins do estudo apresentado, em que a função objetivo levou em conta apenas a redução do caminho percorrido, com a restrição de

que nenhum ponto de operação fosse visitado mais de uma vez, excetuando-se o ponto de retorno em algumas simulações ou experimentos, o algoritmo apresentou resultados satisfatórios, visto que a redução do caminho percorrido foi realmente significativa nos casos estudados. Como pontos positivos do algoritmo genético implementado, podemos citar o poder de redução do caminho percorrido em relação a geração de rota aleatória, versatilidade de se poder entrar com as coordenadas em duas dimensões, mudar facilmente o número da população, número de pontos e o número de iterações, taxa de *crossover*, taxa de mutação, tornando-o adaptável para qualquer problema do tipo em que se deseje encontrar a menor rota possível em um plano 2D, sem restrição de obstáculos. Outro fator que pode ser tido como ponto positivo, é o fato de o algoritmo genético trabalhar *offline*, assim se uma rota fora das restrições impostas ou uma rota longa demais fosse dada como resposta, o processo poderia ser refeito sem por em risco o equipamento. Como pontos negativos podemos citar a dependência do ajuste de parâmetros, custo computacional que cresce mais com o aumento da população. Como a mutação age de forma aleatória, se ocorresse perto das últimas iterações do programa, não haveria tempo de acontecer a convergência e assim uma resposta que poderia ser melhor antes da mutação, era perdida.

O algoritmo de navegação surgiu como uma forma de contornar as restrições impostas por obstáculos, pois no algoritmo genético não havia informações sobre eles. A aplicação das curvas de Bézier para gerar a trajetória local foi uma alternativa satisfatória, visto que o robô não colidiu com os obstáculos e nem desviou muito da trajetória original, tornando a navegação mais segura. Como não houve medição da distância percorrida nas rotas locais, não foi possível obter um número, e assim esta análise é apenas qualitativa, baseada nos experimentos. Como pontos positivos do algoritmo de navegação estão o baixo erro de posição em X e em Y, boa velocidade de contorno de obstáculos (não foi preciso frear o robô), desvio suave, ajuste de parâmetros mais simples de ser realizado (em relação ao ajuste do algoritmo genético), possibilidade de visualização da curva através de softwares gráficos. Como pontos negativos podemos citar o fato de terem sido usados apenas obstáculos cilíndricos, não contabilização da distância percorrida nas rotas locais, o baixo alcance (*range*) dos sensores infravermelhos e os pontos cegos.

Assim, pode-se dizer que o algoritmo de navegação conseguiu cumprir a rota global

estipulado pelo algoritmo genético com sucesso, realizando os devidos ajustes com as curvas de Bézier para desviar dos obstáculos.

5.1 Sugestões para trabalhos futuros

A junção do algoritmo genético e o algoritmo de navegação proporcionaram navegabilidade ao Robotino®. Porém o algoritmo aqui apresentado tem potencial para mais. Por exemplo, fazendo-se as devidas **adaptações** pode ser usado em drones aéreos e também como algoritmo de ultrapassagem para carros autônomos. A seguir, algumas sugestões para trabalhos futuros:

- Expandir o algoritmo genético para a terceira dimensão (espacial).
- Utilizar algoritmo genético para auxiliar a tomada de decisões e ajuste de parâmetros do próprio algoritmo genético e do algortimo de navegação
- Incluir mais restrições no cálculo das rotas no algoritmo genético (distância, tempo, velocidade permitida, etc.)
- Ajustar o momento da mutação.
- Expandir as Curvas de Bézier para terceira dimensão e aumentar a ordem da curva.
- Utilizar o sensor *Laser Range Finder* e expandir a distância de detecção de obstáculos.
- Incluir dinâmicas de ambiente.
- Utilizar câmera para ajudar na detecção de obstáculos e reconhecimento de ambiente.

Tendo em vista que o procedimento aqui discutido pode ser empregado em outras aplicações com as devidas **adaptações**, sugere-se ainda:

- Desenvolver sensor para detecção e desvio de obstáculos para máquinas CNC.
- Criação de um algoritmo de ultrapassagem para veículos autônomos.
- Proporcionar navegabilidade em drones aéreos e ROVs submarinos.

Referências Bibliográficas

Ahmed, Z. H. (2010). Genetic algorithm for the traveling salesman problem using sequential constructive crossover operator. *International Journal of Biometrics & Bioinformatics (IJBB)*, 3(6):96–105.

Avila, S. L. (2002). Algoritmos genéticos aplicados na otimização de antenas refletoras. Dissertação de mestrado, Universidade Federal de Santa Catarina.

Borenstein, J. e Koren, Y. (1991). The vector field histogram - fast obstacle avoidance for mobile robots. *IEEE Journal of Robotics and Automation*, 7:278–288.

Catarina, A. S. e Bach, S. L. (2003). Estudo do efeito dos parâmetros genéticos sobre a solução otimizada e sobre o tempo de convergência em algoritmos genéticos com codificações binária e real. *Acta Scientiarum. Technology*, 25(2):147–152.

Choi, J. W., Curry, R., e Elkaim, G. (2010). Piecewise bézier curves path planning with continuous curvature constraint for autonoumous driving. *Machine Learning and Systems Engineering. Lecture Notes in Electrical Engineering*, pages 31–45.

Choset, H. (2001). Coverage for robotics – a survey of recent results. *Annals of Mathematics and Artificial Intelligence*, 31:113–126.

Cordeiro, F. R. (2008). Uma ferramenta de simulação para otimização multi-objetiva evolucionária. Trabalho de conclusão de curso, Universidade de Pernambuco.

Dorofei, I., Grosu, V., e Spinu, V. (2007). *Omnidirectional Mobile Robot - Design and Implementation*. Bioinspiration and Robotics Walking and Climbing Robots, Maki K. Habib. InTech.

Dwivedi, V., Chauhan, T., Saxena, S., e Agrawal, P. (2012). Travelling salesman problem using genetic algorithm. *National Conference on Development of Reliable Information Systems, Techniques and Related Issues (DRISTI)*, 1:25–30.

Ferreira, A. (2004). Desvio tangencial de obstáculos para um robô móvel navegando em ambientes semi-estruturados. Dissertação de mestrado, UFES.

Festo (2013a). Distance sensor. Disponível em: http://www.festo-didactic.com/int-en/services/robotino/hardware/sensors/distance-sensors/?fbid=aW5OLmVuLjU1Ny4xNy4zNC4xNDUy. Último acesso: 20/10/2014.

Festo (2013b). Laser scanner. Disponível em: http://www.festo-didactic.com/es-es/productos/robotino/sensor-giroscopico-robotino-hasta-2013.htm?fbid=ZXMuZXMuNTQ3LjE0LjE4Ljg1OC43MzI5. Último acesso: 10/11/2014.

Festo (2013c). Omnidrive. Disponível em: http://doc.openrobotino.org/download/RobotinoView/en/index.html?robotino_omnidrive.htm. Último acesso: 10/11/2014.

Fujimori, A., Nikiforuk, P. N., e Gupta, M. M. (1997). Adaptive navigation of mobile robots with obstacle avoidance. *IEEE Transactions on Robotics and Automation*, 13:596–902.

Galceran, E. e Carreras, M. (2013). A survey on coverage path planning for robotics. *Robotics and Autonomous Systems*, 61:1258–1276.

Goldberg, D. E. (1989). *Genetic Algorithms in Search and Optimization and Machine Learning*. Addison Wesley, 1st edition.

Goldberg, D. E. e Lingle, R. (1985). Alleles, loci, and the traveling salesman problem. *Grefenstette JJ (ed) Proceeding of an Intenatiotal Conference on Genetic Algorithms an Their Applications*, pages 154–159.

Hu, H. e Brady, M. (1994). A bayesian approach to real-time obstacle avoidance for a mobile robot. *Autonomous robots*, 1:69–92.

Jolly, K. G., Kumar, R. S., e Vijayakumar, R. (2009). A bezier curve based path planning in a multi-agent robot soccer system without violating the acceleration limits. *Robotics and Autonomous Systems*, 57:23–33.

Kala, R. e Warwick, K. (2014). Heuristic based evolution for the coordination of autonomous vehicles in the absence of speed lanes. *Applied Soft Computing*, 19.

Kwon, K. Y., Cho, J., e Joh, J. (2006). Collision avoidance of moving obstacles for underwater robots. *Systemics, Cybernetics and Informatics*, 4:86–91.

Li, Q., Liu, G., e Yin, Y. (2007). A specialized genetic algorithm for optimum path planning of mobile robots. *Neural, Parallel & Scientific Computations*, 15:279–298.

Lima, E. O. (2008). Algoritmo genético híbrido aplicado à otimização de funções. Trabalho de conclusão de curso, UFES.

Linden, R. (2008). *Algoritmos Genéticos, Uma Importante Ferramenta de Inteligência Computacional*. Brasport, Rio de Janeiro.

Ma, X., Li, X., e Qiao, H. (2001). Fuzzi neural network-based real-time self-reaction of mobile robot in unknown enviroments. *Mechatronics*, 11:1039–1052.

Mitchell, M. (1998). *An Introduction to Genetic Algorithms*. A Bradford book. MIT Press.

Nascimento, T. P. (2009). Controle de trajetória de robôs móveis omni-direcionais: Uma abordagem multivariável. Dissertação de mestrado, Universidade Federal da Bahia.

Oliveira, M. S. (2007). Um estudo sobre algoritmos genéticos. Trabalho de conclusão de curso, UNICEUMA.

Oliveira, M. S., Costa Filho, J., e Araújo, A. L. C. (2012). Projeto e controle de um robô móvel. Disponível em: http://conexoes.ifce.edu.br/index.php/conexoes/article/view/133/124. Último acesso: 08/01/2013.

Pehlivanoglu, Y. V., Baysal, O., e Hacioglu, A. (2007). Path planning for autonomous uav via vibrational genetic algorithm. *Aircraft Engineering and Aerospace Technology: An International Journal*, 79:352–359.

Piratebrine (2013). Thermwood cnc cutting a sketchup bezier curve. Disponível em: https://www.youtube.com/watch?v=7_PPUDi_jgk. Último acesso: 29/09/2015.

Rao, A. e Hedge, S. K. (2015). Literature survey on travelling salesman problem using genetic algorithms. *International Journal of Advanced Research in Eduation Technology (IJARET)*, 2:42–45.

Sahingoz, O. K. (2014). Generation of bezier curve-based flyable trajectories for m ulti-uav systems with p arallel genetic algorithm. *J Intell Robot Syst*, 74:499–511.

Saraiva, F. O. e Oliveira, A. C. (2010). Uma comparação empírica de operadores de crossover para o problema de job shp com datas de entregas. Disponível em: http://www.abepro.org.br/biblioteca/enegep2010_TN_STO_118_772_15277.pdf. Último acesso: 29/10/2014.

Secchi, H. A. (2008). Una introducción a los robots móviles. Monografia, Universidade Nacional de San Juan.

Siciliano, A. V. (2006). Determinação de trajetória Ótima em navegação robótica móvel. Dissertação de mestrado, Universidade Federal do Rio de Janeiro.

Silva, A. F. e Oliveira, A. C. (2006). Algoritmos genéticos: alguns experimentos com os operadores de cruzamento ("crossover") para o problema do caixeiro viajante assimétrico. Disponível em: http://www.abepro.org.br/biblioteca/ENEGEP2006_TR460314_7093.pdf. Último acesso: 29/09/2015.

Sivanandam, S. N. e Deepa, S. N. (2008). *Introduction to Genetic Algorithms*. Springer.

Sivaraj, R. e Ravichandran, T. (2011). A review of selection methods in genetic algorithm. *International Journal of Engineering Science and Technology (IJEST)*, 3:3792–3797.

Skrjank, I. e Klancar, G. (2010). Optimal cooperative collision avoidance between multiple robots based on bernstein–bézier curves. *Robotics and Autonomous Systems*, 58:1–9.

Souza, A. F. (2001). Análise das interpolações de trajetórias de ferramenta na usinagem hsc (high speed cutting) em superfícies complexas. Dissertação de mestrado, UNIMEP.

Von Zuben, F. J. (2011). Computação evolutiva: uma abordagem pragmática. Disponível em: http://www.ic.unicamp.br/~rocha/teaching/2011s2/mc906/aulas/computacao-evolutiva-uma-abordagem-pragmatica.pdf. Último acesso: 28/09/2015.

Weber, R. C. e Bellenberg, M. (2010). *Robotino Manual*. Festo Didactic Gmbh & Co. KG, 73770 Denkendorf, Germany.

Weisstein, E. W. (2009). Bézier curve. Disponível em: http://mathworld.wolfram.com/BezierCurve.html. Último acesso: 11/03/2015.

Apêndice A

Código do Programa

Neste apêndice o código do programa é apresentado

```
1   // constantes matematicas
2  #define _USE_MATH_DEFINES
3   // bibliotecas do c++
4  #include <iostream>
5  #include <cmath>
6  #include <time.h>
7  // bibliotecas do robotino
8  #include "rec/robotino/com/all.h"
9  #include "rec/core_lt/Timer.h"
10 #include "rec/core_lt/utils.h"
11 // utilizacao dos namespaces para reducao do escopo
12 using namespace std;
13 using namespace rec::robotino::com;
14 //variaveis globais
15 int const population=500; //tamanho da populacao de individuos
16 int const cromolength=8;  //tamanho da rota
17 int const nuit=1000;  //numero de iteracoes
18 int nuit2=1000;
19 float medium[nuit]; //calcula a media das distancias em cada populacao
20 int cps=0;
21
22 //CLASSES
23
24 class MyCom : public Com
25 {
26   public:
27     MyCom()
28     {
29     }
30     void errorEvent( Error error, const char* errorString )
```

```
31      {
32        std::cerr << "Error: " << errorString << std::endl;
33      }
34      void connectedEvent()
35      {
36        std::cout << "Connected." << std::endl;
37      }
38      void connectionClosedEvent()
39      {
40        std::cout << "Connection closed." << std::endl;
41      }
42  };
43
44  MyCom com;  //comunicacao
45  OmniDrive omniDrive; //funcoes de velocidade
46  DistanceSensor distancia0; //classe que permite utilizar os sensores infravermelhos
47  DistanceSensor distancia1;
48  DistanceSensor distancia2;
49  DistanceSensor distancia3;
50  DistanceSensor distancia4;
51  DistanceSensor distancia5;
52  DistanceSensor distancia6;
53  DistanceSensor distancia7;
54  DistanceSensor distancia8;
55  ComId comId;
56  Odometry odometry; //funcoes da odometria
57  Motor motor; //funcoes dos motores do robotino
58
59  void init()
60  {
61     omniDrive.setComId( com.id() );
62    // Connect
63    std::cout << "Connecting..." << std::endl;
64    //com.setAddress( "172.26.201.1" ); //endereco do robotino real
65    com.setAddress( "127.0.0.1:8080" ); //endereco para simulador
66    com.connect();
67    std::cout << std::endl << "Connected" << std::endl;
68    odometry.set(0.0f,0.0f,0.0f); //para garantir que o robotino saia da posicao 0
69  }
70
71  void CoordReta(float coordx, float coordy) //movimentacao em linha reta para as
        coordenadas desejadas
72  {
73      float vex; //velocidade em x
74          float vey; //velocidade em y
75          float vel; //velocidade maxima
```

```
76          float ang; //angulo
77
78          vel=200;
79
80          ang=atan2((coordy),(coordx));
81
82          vex=vel*cos(ang);
83          vey=vel*sin(ang);
84          omniDrive.setVelocity(vex, vey, 0.0f);
85
86              if(phi>=5.0 )//correcao da orientacao
87           {
88            motor.setMotorNumber(0);
89            motor.setSpeedSetPoint(-200.0f);
90            motor.setMotorNumber(1);
91            motor.setSpeedSetPoint(-200.0f);
92            motor.setMotorNumber(2);
93            motor.setSpeedSetPoint(-200.0f);
94           }
95           if(phi<=-5.0 )
96           {
97            motor.setMotorNumber(0);
98            motor.setSpeedSetPoint(200.0f);
99            motor.setMotorNumber(1);
100           motor.setSpeedSetPoint(200.0f);
101           motor.setMotorNumber(2);
102           motor.setSpeedSetPoint(200.0f);
103 } //fim da funcao
104
105 int CoordParada(float objx, float objy) //condicao de parada geral
106 {
107         if(abs(objx-odometry.x())<20.0 && abs(objy-odometry.y())<20.0 )
108                         {
109                                         cout<<odometry.phi()<<endl;
110                                         cout<<odometry.x()<<endl;
111                                         cout<<odometry.y()<<endl;
112                                         return 2;
113                         }
114
115 }//fim da funcao
116
117 float estdist(float volts) //estima a distancia do robo para o obstaculo
118 {
119         float distest;
120         distest=(31.828*pow(volts,6) - 321.81*pow(volts,5) + 1325.2*pow(volts, 4) -
              2871.8*pow(volts, 3) + 3530.4*pow(volts,2) - 2463.4*volts + 899.24); //
```

```
            formula estimada
121         return distest;
122 } //fim da funcao
123
124 void DistanceVoltage(float coordx, float coordy, float cx, float cy) //responsavel pela
        deteccao dos obstaculo e pela geracao da rota local
125 {
126         rec::iocontrol::remotestate::SetState setState;
127         rec::iocontrol::remotestate::SensorState sensorState; //habilita o sensor do
                bumper
128         rec::core_lt::Timer timer; //temporizador do robotino
129         float f0, f1, f2, f3, f4, f5, f6, f7, f8;
130         f0 = f1 = f2 = f3 = f4 = f5 = f6 = f7 = f8 = 0;
131         float dir_obj; //direcao do objetivo
132         float thresh=0.6; //threshold
133         float A0, A1, A2, A3, B0, B1, B2, B3; //pontos de controle
134         float dist=400; //distancia do obstaculo que o ponto de controle 1 estara
135         float u=0.0; //numero que varia de 0 a 1 (intervalo em que a curva de bezier e
                definida)
136         int const incre2=16;
137         int incre1=15;
138         float incre=incre1;
139         float objx[incre2]; //coordenada x da curva de bezier
140         float objy[incre2]; //coordenada y da curva de bezier
141         //distancias x e y do obstaculo
142         float qobst0x, qobst0y, qobst1x, qobst1y, qobst2x, qobst2y, qobst3x, qobst3y,
                qobst4x, qobst4y, qobst5x, qobst5y, qobst6x, qobst6y, qobst7x, qobst7y,
                qobst8x, qobst8y;
143         qobst0x = qobst0y = qobst1x = qobst1y = qobst2x = qobst2y = qobst3x = qobst3y =
                qobst4x = qobst4y = qobst5x = qobst5y = qobst6x = qobst6y = qobst7x = qobst7y
                 = qobst8x = qobst8y =0;
144         //direcoes
145         float dir0, dir1, dir2, dir3, dir4, dir5, dir6, dir7, dir8;
146         float distcalc=0;
147
148
149         if(distancia0.voltage()>thresh || distancia1.voltage()>thresh  || distancia2.
                voltage()>thresh || distancia3.voltage()>thresh || distancia4.voltage()>
                thresh || distancia5.voltage()>thresh || distancia6.voltage()>thresh ||
                distancia7.voltage()>thresh || distancia8.voltage()>thresh)
150         {
151
152                 f0 = f1 = f2 = f3 = f4 = f5 = f6 = f7 = f8 = 0;
153                 //calculo da direcao objetivo
154
155                 dir_obj=(atan2((coordy),(coordx))*360.0f)/(2.0f*M_PI);
```

```
156
157                 //cout<<"direcao "<<dir_obj<<endl;
158
159                 //ponto de controle inicial P0(A0,B0)
160                 A0=odometry.x();
161                 B0=odometry.y();
162
163                 //ponto de controle final P3(A3, B3)
164
165                 A3=cx;
166                 B3=cy;
167
168                 //estimacao da distancia para o obstaculo
169
170                 //sensor 0
171                 if(distancia0.voltage()>thresh && ((dir_obj>=0 && dir_obj<40)||(dir_obj
                        >=40 && dir_obj<80)||(dir_obj>=80 && dir_obj<120)||(dir_obj>=-80 &&
                        dir_obj<-40)||(dir_obj>=-40 && dir_obj<0)))
172                 {f0=1;
173                 cout<<"sensor 0: "<<distancia0.voltage()<<endl;}
174
175                 qobst0x = f0*( estdist(distancia0.voltage())*cos(M_PI*0/180));
176
177                 qobst0y = f0*( estdist(distancia0.voltage())*sin(M_PI*0/180));
178
179                 //sensor 1
180                 if(distancia1.voltage()>thresh && ((dir_obj>=0 && dir_obj<40)||(dir_obj
                        >=40 && dir_obj<80)||(dir_obj>=80 && dir_obj<120)||(dir_obj>=120 &&
                        dir_obj<160)||(dir_obj>=-40 && dir_obj<0)))
181                 {f1=1;
182                 cout<<"sensor 1: "<<distancia1.voltage()<<endl;}
183
184                 qobst1x = f1*( estdist(distancia1.voltage())*cos(M_PI*40/180));
185
186                 qobst1y = f1*( estdist(distancia1.voltage())*sin(M_PI*40/180));
187
188                 //sensor 2
189                 if(distancia2.voltage()>thresh && ((dir_obj>=0 && dir_obj<40)||(dir_obj
                        >=40 && dir_obj<80)||(dir_obj>=80 && dir_obj<120)||(dir_obj>=120 &&
                        dir_obj<160)||(dir_obj>=160 && dir_obj<=180 || dir_obj>=-180 &&
                        dir_obj<-160)))
190                 {f2=1;
191                 cout<<"sensor 2: "<<distancia2.voltage()<<endl;}
192
193                 qobst2x = f2*( estdist(distancia2.voltage())*cos(M_PI*80/180));
194
```

```
195                 qobst2y = f2*(estdist(distancia2.voltage())*sin(M_PI*80/180));
196
197                 //sensor 3
198                 if(distancia3.voltage()>thresh && ((dir_obj>=-160 && dir_obj<-120)||(
                        dir_obj>=40 && dir_obj<80)||(dir_obj>=80 && dir_obj<120)||(dir_obj
                        >=120 && dir_obj<160)||(dir_obj>=160 && dir_obj<=180 || dir_obj>=-180
                         && dir_obj<-160)))
199                 {f3=1;
200                 cout<<"sensor 3: "<<distancia3.voltage()<<endl;}
201
202                 qobst3x =f3*(estdist(distancia3.voltage())*cos(M_PI*120/180));
203
204                 qobst3y =f3*( estdist(distancia3.voltage())*sin(M_PI*120/180));
205
206                 //sensor 4
207                 if(distancia4.voltage()>thresh && ((dir_obj>=-160 && dir_obj<-120)||(
                        dir_obj>=-120 && dir_obj<-80)||(dir_obj>=80 && dir_obj<120)||(dir_obj
                        >=120 && dir_obj<160)||(dir_obj>=160 && dir_obj<=180 || dir_obj>=-180
                         && dir_obj<-160)))
208                 {f4=1;
209                 cout<<"sensor 4: "<<distancia4.voltage()<<endl;}
210
211                 qobst4x =f4*( estdist(distancia4.voltage())*cos(M_PI*160/180));
212
213                 qobst4y =f4*( estdist(distancia4.voltage())*sin(M_PI*160/180));
214
215                 //sensor 5
216                 if(distancia5.voltage()>thresh && ((dir_obj>=120 && dir_obj<160)||(
                        dir_obj>=160 && dir_obj<=180 || dir_obj>=-180 && dir_obj<-160)||(
                        dir_obj>=-160 && dir_obj<-120)||(dir_obj>=-120 && dir_obj<-80)||(
                        dir_obj>=-80 && dir_obj<-40)))
217                 {f5=1;
218                 cout<<"sensor 5: "<<distancia5.voltage()<<endl;}
219
220                 qobst5x =f5*(  estdist(distancia5.voltage())*cos(M_PI*-160/180));
221
222                 qobst5y =f5*(  estdist(distancia5.voltage())*sin(M_PI*-160/180));
223
224                 //sensor 6
225                 if(distancia6.voltage()>thresh && ((dir_obj>=-40 && dir_obj<0)||(dir_obj
                        >=160 && dir_obj<=180 || dir_obj>=-180 && dir_obj<-160)||(dir_obj
                        >=-160 && dir_obj<-120)||(dir_obj>=-120 && dir_obj<-80)||(dir_obj
                        >=-80 && dir_obj<-40)))
226                 {f6=1;
227                 cout<<"sensor 6: "<<distancia6.voltage()<<endl;}
228
```

```
229                 qobst6x = f6*(estdist(distancia6.voltage())*cos(M_PI*-120/180));
230
231                 qobst6y = f6*( estdist(distancia6.voltage())*sin(M_PI*-120/180));
232
233                 //sensor 7
234                 if(distancia7.voltage()>thresh && ((dir_obj>=-40 && dir_obj<0)||(dir_obj
                        >=0 && dir_obj<40)||(dir_obj>=-160 && dir_obj<-120)||(dir_obj>=-120
                        && dir_obj<-80)||(dir_obj>=-80 && dir_obj<-40)))
235                 {f7=1;
236                 cout<<"sensor 7: "<<distancia7.voltage()<<endl;}
237
238                 qobst7x = f7*(estdist(distancia7.voltage())*cos(M_PI*-80/180));
239
240                 qobst7y = f7*( estdist(distancia7.voltage())*sin(M_PI*-80/180));
241
242                 //sensor 8
243                 if(distancia8.voltage()>thresh && ((dir_obj>=-40 && dir_obj<0)||(dir_obj
                        >=0 && dir_obj<40)||(dir_obj>=40 && dir_obj<80)||(dir_obj>=-120 &&
                        dir_obj<-80)||(dir_obj>=-80 && dir_obj<-40)))
244                 {f8=1;
245                 cout<<"sensor 8: "<<distancia8.voltage()<<endl;}
246
247                 qobst8x = f8*(estdist(distancia8.voltage())*cos(M_PI*-40/180));
248
249                 qobst8y = f8*(estdist(distancia8.voltage())*sin(M_PI*-40/180));
250
251                         if(dir_obj>=0 && dir_obj<40) //entre os sensores 0 e 1
252                 {
253                         dir0 = -90; //(-90) //centro
254                         dir1 = -50; //(-50)
255                         dir2 = -10; //(170)
256                         dir3 = 30; //(-150)
257                         dir4 = -110;  //(70)
258                         dir5 = -70;   //(110)
259                         dir6 = -30; //(150)
260                         dir7 = 10; //(-170)
261                         dir8 = 50; //(-130)
262                 }
263
264                 if(dir_obj>=40 && dir_obj<80) //entre os sensores 1 e 2
265                 {
266                         dir0 = -90; //(-90)
267                         dir1 = -50; //(-50) //centro
268                         dir2 = -10; //(-170)
269                         dir3 = 30; //(-150)
270                         dir4 = 70;  //(70)
```

```
271                         dir5 = 110;  //(-70)
272                         dir6 = 150;  //(-30)
273                         dir7 = -170; //(-170)
274                         dir8 = -130; //(-130)
275                 }
276
277                 if(dir_obj>=80 && dir_obj<120)//entre os sensores 2 e 3
278                 {
279                         dir0 = 90; //(90)
280                         dir1 = 130; //(130)
281                         dir2 = 170; //(-10) //centro
282                         dir3 = 30; //(-150)
283                         dir4 = 70;  //(70)
284                         dir5 = 110;  //(-70)
285                         dir6 = 150;  //(-30)
286                         dir7 = 10; //(-170)
287                         dir8 = 50; //(-130)
288                 }
289
290                 if(dir_obj>=120 && dir_obj<160)//entre os sensores 3 e 4
291                 {
292                         dir0 = 90; ; //(90)
293                         dir1 = 130; //(-50)
294                         dir2 = 170; //(-10)
295                         dir3 = 30; //(-150) // centro
296                         dir4 = 70;  //(-110)
297                         dir5 = 110;   //(-70)
298                         dir6 = 150; //(-30)
299                         dir7 = -170; //(10)
300                         dir8 = 50; //(-130)
301                 }
302
303                 if(dir_obj>=160 && dir_obj<=180 || dir_obj>=-180 && dir_obj<-160 ) //
                        entre os sensores 4 e 5
304                 {
305                         dir0 = -90; ; //(90)
306                         dir1 = 130; //(130)
307                         dir2 = 170; //(-10)
308                         dir3 = -150; //(30)
309                         dir4 = -110;  //(70) //centro
310                         dir5 = 110;   //(-70)
311                         dir6 = 150; //(-30)
312                         dir7 = -170; //(10)
313                         dir8 = -130; //(50)
314                 }
315
```

```
316                 if(dir_obj>=-160 && dir_obj<-120) //entre os sensores 5 e 6
317                 {
318                         dir0 = -90; ; //(90)
319                         dir1 = -50; //(130)
320                         dir2 = 170; //(-10)
321                         dir3 = -150; //(30)
322                         dir4 = -110;  //(-110)
323                         dir5 = -70;   //(-70) //centro
324                         dir6 = -30; //(-30)
325                         dir7 = 10; //(10)
326                         dir8 = 50; //(50)
327                 }
328
329                 if(dir_obj>=-120 && dir_obj<-80) //entre os sensores 6 e 7
330                 {
331                         dir0 = 90; ; //(90)
332                         dir1 = 130; //(130)
333                         dir2 = 170; //(170)
334                         dir3 = -150; //(30)
335                         dir4 = -110;  //(70)
336                         dir5 = -70;   //(110)
337                         dir6 = -30; //(-30) //centro
338                         dir7 = 10; //(10)
339                         dir8 = 50; //(50)
340                 }
341
342                 if(dir_obj>=-80 && dir_obj<-40) //entre os sensores 7 e 8
343                 {
344                         dir0 = -90; ; //(90)
345                         dir1 = -50; //(130)
346                         dir2 = -10; //(170)
347                         dir3 = 30; //(-150)
348                         dir4 = 70;  //(-110)
349                         dir5 = -70;   //(110)
350                         dir6 = -30; //(150)
351                         dir7 = 10; //(-170) //centro
352                         dir8 = 50; //(-130)
353                 }
354
355                 if(dir_obj>=-40 && dir_obj<0) //entre os sensores 8 e 0
356                 {
357                         dir0 = -90; ; //(90)
358                         dir1 = -50; //(130)
359                         dir2 = -10; //(170)
360                         dir3 = 30; //(-150)
361                         dir4 = -110;  //(70)
```

```
362                         dir5 = -70;   //(110)
363                         dir6 = -30; //(-30)
364                         dir7 = 10; //(-170)
365                         dir8 = 50; //(-130) //centro
366                 }
367
368 if(cps==3) //paralelo a normal
369                 {
370
371                 if(dir_obj>=0 && dir_obj<40) //entre os sensores 0 e 1
372                 {
373                         dir0 = -180; //(-90) //centro
374                         dir1 = -140; //(-50)
375                         dir2 = -100; //(170)
376                         dir3 = -60; //(-150)
377                         dir4 = -20;  //(70)
378                         dir5 = 20;   //(110)
379                         dir6 = 60; //(150)
380                         dir7 = 100; //(-170)
381                         dir8 = 140; //(-130)
382                 }
383
384                 if(dir_obj>=40 && dir_obj<80) //entre os sensores 1 e 2
385                 {
386                         dir0 = -180; //(-90) //centro
387                         dir1 = -140; //(-50)
388                         dir2 = -100; //(170)
389                         dir3 = -60; //(-150)
390                         dir4 = -20;  //(70)
391                         dir5 = 20;   //(110)
392                         dir6 = 60; //(150)
393                         dir7 = 100; //(-170)
394                         dir8 = 140; //(-130)
395                 }
396
397                 if(dir_obj>=80 && dir_obj<120)//entre os sensores 2 e 3
398                 {
399                         dir0 = -180; //(-90) //centro
400                         dir1 = -140; //(-50)
401                         dir2 = -100; //(170)
402                         dir3 = -60; //(-150)
403                         dir4 = -20;  //(70)
404                         dir5 = 20;   //(110)
405                         dir6 = 60; //(150)
406                         dir7 = 100; //(-170)
407                         dir8 = 140; //(-130)
```

```
408                 }
409
410                 if(dir_obj>=120 && dir_obj<160)//entre os sensores 3 e 4
411                 {
412                         dir0 = -180; //(-90) //centro
413                         dir1 = -140; //(-50)
414                         dir2 = -100; //(170)
415                         dir3 = -60; //(-150)
416                         dir4 = -20;  //(70)
417                         dir5 = 20;   //(110)
418                         dir6 = 60; //(150)
419                         dir7 = 100; //(-170)
420                         dir8 = 140; //(-130)
421                 }
422
423                 if(dir_obj>=160 && dir_obj<=180 || dir_obj>=-180 && dir_obj<-160 ) //
                        entre os sensores 4 e 5
424                 {
425                         dir0 = -180; //(-90) //centro
426                         dir1 = -140; //(-50)
427                         dir2 = -100; //(170)
428                         dir3 = -60; //(-150)
429                         dir4 = -20;  //(70)
430                         dir5 = 20;   //(110)
431                         dir6 = 60; //(150)
432                         dir7 = 100; //(-170)
433                         dir8 = 140; //(-130)
434                 }
435
436                 if(dir_obj>=-160 && dir_obj<-120) //entre os sensores 5 e 6
437                 {
438                         dir0 = -180; //(-90) //centro
439                         dir1 = -140; //(-50)
440                         dir2 = -100; //(170)
441                         dir3 = -60; //(-150)
442                         dir4 = -20;  //(70)
443                         dir5 = 20;   //(110)
444                         dir6 = 60; //(150)
445                         dir7 = 100; //(-170)
446                         dir8 = 140; //(-130)
447                 }
448
449                 if(dir_obj>=-120 && dir_obj<-80) //entre os sensores 6 e 7
450                 {
451                         dir0 = -180; //(-90) //centro
452                         dir1 = -140; //(-50)
```

```
453                         dir2 = -100; //(170)
454                         dir3 = -60; //(-150)
455                         dir4 = -20;  //(70)
456                         dir5 = 20;   //(110)
457                         dir6 = 60; //(150)
458                         dir7 = 100; //(-170)
459                         dir8 = 140; //(-130)
460                 }
461
462                 if(dir_obj>=-80 && dir_obj<-40) //entre os sensores 7 e 8
463                 {
464                         dir0 = -180; //(-90) //centro
465                         dir1 = -140; //(-50)
466                         dir2 = -100; //(170)
467                         dir3 = -60; //(-150)
468                         dir4 = -20;  //(70)
469                         dir5 = 20;   //(110)
470                         dir6 = 60; //(150)
471                         dir7 = 100; //(-170)
472                         dir8 = 140; //(-130)
473                 }
474
475                 if(dir_obj>=-40 && dir_obj<0) //entre os sensores 8 e 0
476                 {
477                         dir0 = -180; //(-90) //centro
478                         dir1 = -140; //(-50)
479                         dir2 = -100; //(170)
480                         dir3 = -60; //(-150)
481                         dir4 = -20;  //(70)
482                         dir5 = 20;   //(110)
483                         dir6 = 60; //(150)
484                         dir7 = 100; //(-170)
485                         dir8 = 140; //(-130)
486                 }
487         //
488                 }
489
490                 //calculo do A1 e B1
491                 A1= A0 + qobst0x + f0*dist*cos(M_PI*dir0/180) + qobst1x + f1*dist*cos(
                        M_PI*dir1/180) + qobst2x + f2*dist*cos(M_PI*dir2/180) + qobst3x + f3*
                        dist*cos(M_PI*dir3/180) + qobst4x + f4*dist*cos(M_PI*dir4/180) +
                        qobst5x + f5*dist*cos(M_PI*dir5/180) + qobst6x + f6*dist*cos(M_PI*
                        dir6/180) + qobst7x + f7*dist*cos(M_PI*dir7/180) + qobst8x + f8*dist*
                        cos(M_PI*dir8/180);
492
```

```
493                 B1= B0 + qobst0y + f0*dist*sin(M_PI*dir0/180) + qobst1y + f1*dist*sin(
                        M_PI*dir1/180) + qobst2y + f2*dist*sin(M_PI*dir2/180) + qobst3y + f3*
                        dist*sin(M_PI*dir3/180) + qobst4y + f4*dist*sin(M_PI*dir4/180) +
                        qobst5y + f5*dist*sin(M_PI*dir5/180) + qobst6y + f6*dist*sin(M_PI*
                        dir6/180) + qobst7y + f7*dist*sin(M_PI*dir7/180) + qobst8y + f8*dist*
                        sin(M_PI*dir8/180);
494
495                 //Calculo do A2 e B2
496
497                 A2=(A1 + A3)/2;
498
499                 B2=(B1 + B3)/2;
500
501                 cps=0;
502
503                 //calculo de x e y
504                 int cp=0; //auxiliar break
505                 float thresh2=0.9; //segundo threshold
506                 float thresh3=1.4; //terceiro threshold
507                 for(int i=0; i<=incre1;i++) //lembrar de ajustar o numero de pontos
508                 {cp=0;
509                 objx[i]= A0 + u*(-3*A0+3*A1) + (pow(u, 2))*(3*A0 - 6*A1 + 3*A2) + (pow(u,
                        3))*(-A0 + 3*A1 - 3*A2 + A3);
510
511                 objy[i]= B0 + u*(-3*B0+3*B1) + (pow(u, 2))*(3*B0 - 6*B1 + 3*B2) + (pow(u,
                        3))*(-B0 + 3*B1 - 3*B2 + B3);
512
513                 if(i==incre1)
514                 {cout<<"ponto"<<endl;
515                 }
516
517                 while(CoordParada(objx[i], objy[i])!=2) //se tirar o while, fica
                        recalculando toda hora
518         {
519          CoordReta(objx[i]-odometry.x(),objy[i]-odometry.y(), odometry.phi()); //indica
                as velocidades
520
521     if((distancia0.voltage()>thresh2 && distancia0.voltage()<thresh3) || (distancia1.
        voltage()>thresh2 && distancia1.voltage()<thresh3)  || (distancia2.voltage()>
        thresh2 && distancia2.voltage()<thresh3) || (distancia3.voltage()>thresh2 &&
        distancia3.voltage()<thresh3) || (distancia4.voltage()>thresh2 && distancia4.
        voltage()<thresh3) || (distancia5.voltage()>thresh2 && distancia5.voltage()<
        thresh3) || (distancia6.voltage()>thresh2 && distancia6.voltage()<thresh3) || (
        distancia7.voltage()>thresh2 && distancia7.voltage()<thresh3) || (distancia8.
        voltage()>thresh2 && distancia8.voltage()<thresh3))
522             {
```

```
523                         cp=2;
524                 break;
525                   }
526
527           if((distancia0.voltage()>thresh3) || (distancia1.voltage()>thresh3)  || (
                  distancia2.voltage()>thresh3) || (distancia3.voltage()>thresh3) || (
                  distancia4.voltage()>thresh3) || (distancia5.voltage()>thresh3) || (
                  distancia6.voltage()>thresh3) || (distancia7.voltage()>thresh3) || (
                  distancia8.voltage()>thresh3))
528              {
529                         cp=3;
530                         cps=cp;
531                 break;
532                   }
533
534           u=u+1/incre;
535                   if(cp==2)
536                   {
537                            break;
538                   }
539
540                 } //fim do for
541
542         } //fim do if
543
544 } //fim da funcao
545
546 void drive()
547 {
548   rec::core_lt::Timer timer;     //possibilita o uso de tempo
549   rec::iocontrol::remotestate::SensorState sensorState; //habilita o sensor do bumper
550   rec::iocontrol::remotestate::SetState setState;
551
552   //ALGORITMO GENETICO
553
554 float cx[cromolength];
555 float cy[cromolength];
556 srand((int) time(NULL));
557 unsigned int rota[population][cromolength];
558 int pop, flag;
559 int contmut=0;
560
561 for(int k=0;k<population;k++) //(CRIACAO DAS ROTAS INICIAIS)
562 {
563 rota[k][0]=0; //Representa que sai do ponto 0
564 rota[k][cromolength-1]=cromolength-1;//Representa que volta para o ponto 0
```

```
565 for(int i=1;i<cromolength-1;i++) //-1
566 {
567 pop=rand()$\%$(cromolength-1); //Gera os numeros randomicos -1
568 flag=0; //Artificio
569 for(int j=0;j<i;j++)
570 {
571 if(rota[k][j]==pop) //Condicao para nao repetir numeros
572 {
573 flag=1;
574 break; //Para o loop e vai proximo codigo
575 }
576 }
577 if(flag==1)
578 {i=i-1; //Para garantir a saida do numero certo de alelos
579 continue; //Comeca um novo loop
580 }
581 rota[k][i]=pop;
582 }
583 }
584 cout<<endl;
585
586 //ENTRADA DAS COORDENADAS X E Y
587
588 float P[cromolength][2]; //matriz de coordenadas
589 cout<<"Entre com as coordenadas x "<<endl;
590 for(int i=0;i<cromolength;i++)
591 {
592 cin>>P[i][0]; //Entrada das coordenadas X
593 }
594
595 cout<<"Entre com as coordenadas y "<<endl;
596 for(int i=0;i<cromolength;i++)
597 {
598 cin>>P[i][1]; //Entrada das coordenadas Y
599 }
600 cout<<endl<<endl;
601
602 for(int super=0; super<nuit; super++) //grande laco
603 {
604 float d[population][cromolength-1]; //Distancias de cidade para cidade
605 float somatorio_rota[population]; //Distancias de cada rota (cromossomo)
606 float somatorio_total=0; //Somatorio de todos as aptidoes da populacao
607 float media=0;
608 for(int i=0; i<population; i++) //inicializacao dos vetores
609 {
610 somatorio_rota[i]=0;
```

```
611 }
612
613 //CALCULO DAS DISTANCIAS
614 for(int k=0; k<population; k++)
615 {
616 for(int j=0; j<cromolength-1; j++)
617 {
618         d[k][j]=sqrt(pow(P[rota[k][j]][0]-P[rota[k][j+1]][0],2)+pow(P[rota[k][j]][1]-P[
                rota[k][j+1]][1],2)); //distancia entre os pontos
619         somatorio_rota[k]=somatorio_rota[k]+d[k][j];
620           somatorio_total=somatorio_total+d[k][j];
621         media=somatorio_total/population;
622 }
623 }
624
625 medium[super]=media;
626
627 //PROBABILIDADE DE SELECAO
628 float probi[population]; //Probabilidade
629 float exp_count[population]; //Contagem esperada
630 float percent[population]; //Porcentagem
631
632 for(int i=0;i<population;i++)
633 {
634 probi[i]=1-((somatorio_rota[i])/(somatorio_total)); //"1-..."Adaptacao para a
        minimizacao.
635 percent[i]=probi[i]*100;
636 exp_count[i]=1-((somatorio_rota[i])/(somatorio_total/population)); //Da mesma forma.
637 }
638 //Declaracao de mais variaveis auxiliares
639 int  min1=somatorio_rota[0], min2=somatorio_rota[0], i,max1=somatorio_rota[0], max2=
        somatorio_rota[0];
640  int pos[2]; //posicoes de minimo
641  pos[0]=0;
642  pos[1]=0;
643  int posmax[2]; //posicoes de maximo
644  posmax[0]=0;
645  posmax[1]=0;
646 //Calculando a posicao do minimo
647  for(i=0;i<population;i++)
648  {
649   if(somatorio_rota[i]<min1)
650   {
651    min1=somatorio_rota[i];
652    pos[0]=i;
653   }
```

```
654  }
655 //Calculando a posicao do segundo minimo
656  for(i=0;i<population;i++)
657  {
658   if(somatorio_rota[i]<min2&&i!=pos[0])
659   {
660    min2=somatorio_rota[i];
661    pos[1]=i;
662   }
663  }
664  //Calculando a posicao de maximo
665  for(i=0;i<population;i++)
666  {
667   if(somatorio_rota[i]>max1)
668   {
669    max1=somatorio_rota[i];
670    posmax[0]=i;
671   }
672  }
673 //Calculando o segundo maximo
674  for(i=0;i<population;i++)
675  {
676   if(somatorio_rota[i]>max2&&i!=posmax[0])
677   {
678    max2=somatorio_rota[i];
679    posmax[1]=i;
680   }
681  }
682
683 //SELECAO DOS PONTOS DE CROSSOVER
684
685 int crosspt1, crosspt2; //Pontos de crossover
686 int pais[2][cromolength]; //Pais
687 int tempo1[2][cromolength], tempo2, tempo, cnt,j; //variaveis auxiliares
688 cnt=0;
689
690 do
691  {
692   crosspt1=rand()$\%$cromolength; //Selecao dos pontos de crossover de forma aleatoria
693  }while(crosspt1>2) ;
694  do
695  {
696   crosspt2=rand()$\%$cromolength; //Selecao dos pontos de crossover de forma aleatoria
697  }while(crosspt2<=3);
698  for(j=0;j<cromolength;j++)
699  {
```

```
700   pais[0][j]=rota[pos[0]][j]; //Pai 1, melhor individuo da populacao
701  }
702  for(j=0;j<cromolength;j++)
703  {
704   pais[1][j]=rota[pos[1]][j]; //Pai 2, segundo melhor individuo da populacao
705  }
706  for(int j=crosspt1+1;j<=crosspt2;j++)
707  { cnt++;
708   tempo1[1][cnt]=pais[0][j];
709   tempo1[0][cnt]=pais[1][j];
710   tempo=pais[0][j];
711   pais[0][j]=pais[1][j];
712   pais[1][j]=tempo;
713 }
714
715 //Crossover PMX
716  int numerex;
717 numerex=rand()$\%$4;
718 if(numerex==0)
719 {
720 int k,m; //Variaveis auxiliares
721  for(m=0;m<2;m++)
722  {
723   for(i=0;i<crosspt1+1;i++) //Analisando a rota antes do crosspt 1
724   {
725    for(j=0;j<cnt;j++) //Comparando a rota dentro dos pontos de crossover
726    {
727     if(pais[m][i]==tempo1[m][j])
728     {
729      if(m==0) //Para crianca 1
730      {
731       tempo2=tempo1[1][j]; //Pegar a rota da crianca 2
732       for(k=0;k<cromolength;k++)
733       {
734        if(pais[m][k]==tempo2)  //Se  houver  erro, repita o processo
735        { tempo2=pais[1][k];
736        k=0;
737        }
738       }
739       pais[m][i]=tempo2;  //Atribuindo  os  valores a crianca
740      }
741      else //Para a crianca 2
742      {
743       tempo2=tempo1[0][j];
744       for(k=0;k<cromolength;k++)
745       {
```

```
746         if(pais[m][k]==tempo2) //Se  houver  erro, repita o processo
747         {tempo2=pais[0][k];
748         k=0;
749             }
750        }
751        pais[m][i]=tempo2;  //Atribuindo  os  valores  a crianca
752       }
753      }
754    }
755   }
756  }
757  for(m=0;m<2;m++)
758  {
759   for(i=crosspt2+1;i<cromolength;i++) //Checando a rota depois do crosspt 2
760   {
761    for(j=0;j<cnt;j++) //Comparando a rota dentro dos pontos de crossover
762    {
763     if(pais[m][i]==tempo1[m][j])
764     {
765      if(m==0) //para crianca 1
766      {
767       tempo2=tempo1[1][j]; //Pegar a rota da crianca 2
768       for(k=0;k<cromolength;k++)
769       {
770        if(pais[m][k]==tempo2) //Se  houver  erro, repita o processo
771        {tempo2=pais[1][k];
772        k=0;
773        }
774       }
775       pais[m][i]=tempo2; //Atribuindo  os  valores  a crianca
776      }
777      else //para crianca 2
778      {
779
780       tempo2=tempo1[0][j];
781       for(k=0;k<cnt;k++)
782       {
783        if(pais[m][k]==tempo2) //Se  houver  erro, repita o processo
784        {tempo2=pais[0][k];
785        k=0;
786        }
787       }
788       pais[m][i]=tempo2; //Atribuindo  os  valores  a crianca
789      }
790     }
791    }
```

```
792   }
793  }
794 }
795
796 //MUTACAO
797
798 int numero, numero1, numero2;
799 float paitemp;
800 int ajuda=1;
801
802 numero=rand()$\%$100;
803 if(numero==9)
804 {       //Probabilidade de ocorrencia de 1%
805         contmut++;
806 cout<<"mutacao"<<endl;
807 numero1=rand()$\%$cromolength;
808 numero2=rand()$\%$cromolength;
809 while(numero1==numero2||numero1==0||numero2==0)
810 {
811 numero1=rand()$\%$cromolength;
812 numero2=rand()$\%$cromolength;
813 }
814 paitemp=pais[i][numero1];
815 pais[i][numero1]=pais[i][numero2];//Individuo mutado
816 pais[i][numero2]=paitemp;
817 }
818
819 //Substituicao: Inclusao das criancas no lugar dos piores individuos
820 for(int j=0;j<cromolength;j++)
821  {
822   rota[posmax[0]][j]=pais[0][j];
823   rota[posmax[1]][j]=pais[1][j];
824  }
825 }
826
827 //Apresentacao dos parametros para comparacao
828
829 cout<<"pontos x"<<endl;
830 for(int i=population-50; i<population; i++)
831 { cout<<"individuo "<<i<<"-----";
832 for(int j=0;j<cromolength;j++)
833  {
834  cout<<P[rota[i][j]][0]<<"-"; //pontos x
835  }
836 cout<<endl;
837 }
```

```
838
839 cout<<"pontos y"<<endl;
840 for(int i=population-50; i<population; i++)
841 { cout<<"individuo "<<i<<"-----";
842 for(int j=0;j<cromolength;j++)
843  {
844  cout<<P[rota[i][j]][1]<<"-"; //Pontos y
845  }
846 cout<<endl; cout<<endl;
847 }
848
849 cout<<"Media das distancias "<<endl<<" Populacao inicial "<<medium[0]<<endl;
850
851 cout<<"Populacao final "<<medium[nuit-1]<<endl;
852
853 cout<<endl<<"Decrescimo  relativo  "<<((medium[0]-medium[nuit-1])/medium[0])*100<<endl;
854
855 cout<<"Mutacoes = "<<contmut<<endl;
856 int alea=0;
857 cout<<"entre com o o numero do individuo desejado: ";
858 cin>>alea;
859
860  //individuo aleatorio da ultima populacao
861 cout<<"populacao "<<alea<<endl<<endl;
862
863 for(int j=0;j<cromolength;j++) //apresentacao da rota escolhida
864  {
865  cout<<rota[alea][j]<<" || ";
866  }
867 cout<<endl<<endl;
868 for(int j=0;j<cromolength;j++) //atribuicao dos pontos x
869  {
870  cx[j]=P[(rota[alea][j])][0];
871  }
872
873 for(int j=0;j<cromolength;j++) //atribuicao dos pontos y
874  {
875  cy[j]=P[(rota[alea][j])][1];
876  }
877
878   timer.start();
                //inicializa o tempo
879   for(int i=0; i<cromolength;i++) //quantidade de pontos
880   {
881
882    while( com.isConnected() && false == sensorState.bumper )
```

```
883   {
884         CoordReta(cx[i]-odometry.x(),cy[i]-odometry.y(), odometry.phi());
              //indica as velocidades
885
886         if(i==0) //coordenada zero
887         {  DistanceVoltage(cx[i], cy[i], cx[i], cy[i]); }//desvio de obstaculo
888         else
889         {DistanceVoltage(cx[i]-odometry.x(), cy[i]-odometry.y(), cx[i], cy[i]); }
890
891          if(CoordParada(cx[i], cy[i])==2)
892          {
893                   break;
894          }
895
896         com.waitForUpdate(); //wait until actor set values are transmitted and new sensor
               readings are available
897   }
898   }
899 }//fim da funcao
900
901 void destroy()
902 {
903   com.disconnect(); //disconecta o robotino
904 }//fim da funcao
905
906 int main() //funcao principal
907 {
908   try
909   {     //inicializa o robotino
910         init();
911
912                 //liga os sensores infravermelhos
913                  distancia0.setComId(com.id());
914                 distancia0.setSensorNumber(0);
915
916                 distancia1.setComId(com.id());
917                 distancia1.setSensorNumber(1);
918
919                 distancia2.setComId(com.id());
920                 distancia2.setSensorNumber(2);
921
922                 distancia3.setComId(com.id());
923                 distancia3.setSensorNumber(3);
924
925                 distancia4.setComId(com.id());
926                 distancia4.setSensorNumber(4);
```

```
927
928                 distancia5.setComId(com.id());
929                 distancia5.setSensorNumber(5);
930
931                 distancia6.setComId(com.id());
932                 distancia6.setSensorNumber(6);
933
934                 distancia7.setComId(com.id());
935                 distancia7.setSensorNumber(7);
936
937                 distancia8.setComId(com.id());
938                 distancia8.setSensorNumber(8);
939
940                 drive(); //funcao de navegacao
941
942                 destroy(); //disconecta o robotino
943   }
944   catch( const std::exception& e )
945   {
946     std::cerr << "Error: " << e.what() << std::endl;
947   }
948   std::cout << "Press any key to exit..." << std::endl;
949   rec::core_lt::waitForKey();
950   return 0;
951 } //fim de programa
```