UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

CÁLCULO DE ODOMETRIA DE UM ROBÔ LEGO MINDSTORMS ATRAVÉS DE REDE NEURAL

FERNANDA C. S. RODRIGUES

BLUMENAU
2014

2014/2-08

FERNANDA C. S. RODRIGUES

CÁLCULO DE ODOMETRIA DE UM ROBÔ LEGO MINDSTORMS ATRAVÉS DE REDE NEURAL

Trabalho de Conclusão de Curso apresentado ao curso de graduação em Ciência da Computação do Centro de Ciências Exatas e Naturais da Universidade Regional de Blumenau como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

Prof. Roberto Heinzle, Doutor – Orientador

BLUMENAU
2014

2014/2-08

CÁLCULO DE ODOMETRIA DE UM ROBÔ LEGO MINDSTORMS ATRAVÉS DE REDE NEURAL

Por

FERNANDA C. S. RODRIGUES

Trabalho de Conclusão de Curso aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II pela banca examinadora formada por:

Presidente:	Prof. Roberto Heinzle, Doutor – Orientador, FURB
Membro:	Prof. Mauro Marcelo Mattos, Doutor – FURB
Membro:	Prof. Miguel Alexandre Wisintainer, Mestre – FURB

Blumenau, 12 de dezembro de 2014

Dedico este trabalho aos meus pais e à minha irmã, Flávia. Ao meu namorado, Augusto e à minha amiga, Érika. Todos, cada um ao seu modo, contribuíram para as minhas escolhas e para o que sou.

AGRADECIMENTOS

Aos meus pais por terem me mostrado, desde criança, que o estudo é o melhor caminho para alcançar meus objetivos. À minha irmã, Flávia, que sempre me acompanhou pelas diferentes fases de nossas vidas e com quem aprendi muito.

Ao meu namorado, Augusto, por ter compreendido minhas inúmeras ausências e por ter contribuído, questionando e apoiando incondicionalmente, todas as minhas escolhas. À minha amiga, Érika, que participou de inúmeras das minhas decisões e por ter sempre me incentivado e colaborado no meu desenvolvimento pessoal. Aos meus amigos Maria, Renato e Leandro por todo o tempo que discutimos sobre o TCC e sobre a vida.

Ao meu orientador Roberto Heinzle por ter me aceitado como orientanda, mesmo que o assunto não estivesse completamente relacionado à sua área de atuação.

Não posso deixar de agradecer aos professores Eleri Cardozo, da Unicamp, e Edson Prestes, da UFRGS, que me auxiliaram no andamento deste trabalho sem mesmo me conhecer. Por fim, agradeço à professora Joyce Martins, e aos professores Dalton Solano dos Reis, Aurélio Faustino Hoppe e Maurício Capobianco Lopes, da FURB, que me demonstraram, através das aulas ministradas e das longas conversas, o comprometimento, o fascínio, a importância e a riqueza da carreira acadêmica.

Tenha em mente que tudo que você aprende na escola é trabalho de muitas gerações. Receba essa herança, honre-a, acrescente a ela e, um dia, fielmente, deposite-a nas mãos de seus filhos.

Albert Einstein

RESUMO

Uma das problemáticas encontradas no âmbito da navegação robótica autônoma é a capacidade do próprio sistema em estimar sua posição. Ao estimar a posição o robô tem a possibilidade de verificar se está no ponto correto da trajetória percorrida, possibilitando eventuais correções. Uma das formas do sistema robótico estimar a própria posição é quando ele parte de um ponto conhecido e, a partir dos valores de odometria, estima sua nova posição. Ocorre que por vezes existem defeitos na estrutura do robô que são ignorados pelo cálculo de odometria ocasionando o acúmulo ilimitado de erros de estimação de posição. Este trabalho apresenta a implementação do cálculo de odometria e do método de Fusão de Sensores Rede Neural de Função Radial como alternativa de estimação de posição. O método Rede Neural de Função Radial une os valores da odometria do robô, baseado na plataforma LEGO Mindstorms, e de uma bússola acoplada a ele, em auxílio ao cálculo da posição estimada. A vantagem do uso do método de Rede Neural de Função Radial em relação ao cálculo de odometria é verificada através da comparação entre a distância da posição estimada e a posição real do robô para cada um dos métodos implementados.

Palavras-chave: Odometria. Rede neural de função radial. Fusão de sensores. Comparação de métodos.

ABSTRACT

One of the problems encountered in the context of autonomous robotic navigation its the ability of the system to estimate its own position. Thus, the robot has the possibility to check if his position is in the correct point of the traveled trajectory, allowing possible fixes. One of the forms for the robotic system to estimate his own position is when he leaves from of a known point and, from odometry values, estimates its new position. However, sometimes there are defects in the structure of the robot that are ignored by the odometry calculation, causing the unlimited accumulation of position estimation errors. This work presents the implementation of the odometry calculation and the Radial Function Neural Network Sensor Fusion method of as an alternative position estimation. The Radial Function Neural Network method combines the values of the odometer of the robot, based on LEGO Mindstorms platform, and a compass coupled to it, aiding the calculation of the estimated position. The advantage of using the Radial Function Neural Network in relation to odometry calculation is verified by comparing the distance of the estimated position and the actual position of the robot for each of the implemented methods.

Key-words: Odometry. Radial basis neural network. Sensor fusion. Comparison of methods.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE SIGLAS

API – *Application Programming Interface*

EA – Enterprise Architect

DR – *Dead Reckoning*

FS – Fusão de sensores

IDE – *Integrated Development Environment*

LCD – *Liquid Crystal Display*

UML – *Unified Modeling Language*

RNA – Rede Neural Artificial

RNFR – Rede Neural de Função Radial

LISTA DE SÍMBOLOS

Θ – Valor correspondente à orientação do robô

Δ – Representa a variação de determinado valor

Π – Representa a constante matemática de valor 3,1415

T – Valor que indica o intervalo de tempo

β – Parâmetro de correção do erro sistemático

SUMÁRIO

1 INTRODUÇÃO

Com os atuais avanços da tecnologia é crescente o interesse pelo desenvolvimento da área da robótica por parte de empresas, estudiosos e público consumidor em geral. Um dos motivos se dá pela possibilidade de os robôs executarem certas tarefas com maior eficiência, eficácia e segurança que um humano. Para muitas dessas atividades é essencial que o robô possua autonomia com pouca ou nenhuma interferência humana (COUTO, 2012, p. 1).

O desenvolvimento do estudo de robôs autônomos se deve à diversidade de tarefas em que é possível aplicá-los. Podem ser utilizados na movimentação independente de automóveis pelas estradas; na criação de veículos aéreos não tripulados em auxílio à pulverização de lavouras ou operações militares; como veículos autônomos subaquáticos para realização de explorações no fundo do mar (RUSSEL; NORVIG, 2004, p. 871; SANDI, HEMERLY; LAGES, 1998, p. 107), entre outras aplicações.

Os robôs autônomos operam com base em sistemas próprios de controle, nos quais existem duas problemáticas principais. A primeira é em relação à navegação, que representa a definição da posição e orientação do robô no ambiente em determinado tempo. A segunda é em relação à guiagem robótica que se refere ao controle de sua trajetória. Estas dificuldades existem porque há a dependência da precisão e da natureza dos sensores utilizados (SANDI, HEMERLY; LAGES, 1998, p. 107; PETROVIĆ; IVANJKO; PERIĆ, 2002, p. 91).

Em geral os sensores são imprecisos e as situações às quais são expostos são dinâmicas e imprevisíveis (RUSSEL; NORVIG, 2004, p. 875). Por consequência, as leituras dos valores do ambiente culminam no aparecimento de erros de medição. Como alternativa, é possível arranjar os sensores disponíveis a fim de auxiliar na redução desses erros. A combinação das informações garante que limitações apresentadas por um sensor possam ser compensadas pelos demais sensores existentes. A união desses dados é em geral realizada através da Fusão de Sensores (FS) e permite obter uma informação mais completa, precisa e confiável. A FS possibilita adquirir características do ambiente que ultrapassam a capacidade individual dos sensores (SANDI, HEMERLY; LAGES, 1998, p. 107; PETROVIĆ; IVANJKO; PERIĆ, 2002, p. 91; FACELI, 2001, p. 3).

Dentro da tarefa da navegabilidade robótica um dos métodos mais utilizados para estimar a posição momentânea de um robô a partir de um ponto conhecido é a odometria. A odometria representa o valor da distância percorrida por um robô e é adquirida a partir do sensor que registra a revolução do motor. No entanto, este valor pode estar incorreto pois podem ocorrer derrapagens, existir rugosidades no chão, rodas com medidas diferentes, entre

outros problemas. Por conta disso comumente a odometria é combinada com o valor da bússola através da FS. Por intermédio do uso da bússola é possível utilizar as medidas de orientação em auxílio à navegabilidade robótica (PETROVIĆ; IVANJKO; PERIĆ, 2002, p. 91; SANDI, HEMERLY; LAGES, 1998, p. 107).

Tendo em vista a vantagem do uso da odometria combinada com a bússola, este trabalho apresentada a implementação do método de FS Rede Neural de Função Radial (RNFR) aplicado à navegabilidade de um robô LEGO Mindstorms. Este método é utilizado com o propósito de auxiliar na estimação da posição e orientação robótica. A partir de sua execução são demonstradas as correções de guiagem necessárias para percorrer determinadas trajetórias de interesse. Como resultado do trabalho, é apresentado um comparativo entre a implementação da navegação utilizando o método RNFR em relação à adoção apenas do cálculo de odometria. A comparação é realizada através da posição estimada por intermédio de cada implementação e a posição real do robô no ambiente. Através dessa comparação é possível verificar qual implementação se mostra mais adequada, uma vez que quanto menor a diferença da posição e orientação estimada em relação a posição e orientação real, mais precisa é a navegabilidade e consequentemente a guiagem do robô.

1.1 OBJETIVOS

O objetivo desse trabalho é disponibilizar o cálculo de odometria auxiliado pelo método de FS Rede Neural de Fusão Radial em um robô LEGO Mindstorms.

Os objetivos específicos do trabalho são:

a) criar um robô na plataforma LEGO Mindstorms provido de dois sensores odométricos e uma bússola;
b) disponibilizar um protótipo para coleta e exibição gráfica dos dados de execução dos métodos implementados;
c) comparar o resultado obtido com o presente trabalho em relação aos correlatos.

1.2 ESTRUTURA

Este trabalho possui quatro capítulos: introdução, fundamentação teórica, desenvolvimento e a apresentação das discussões e resultados. O capítulo de introdução descreve a problemática da navegação de robôs autônomos seguido das justificativas para o desenvolvimento deste trabalho. O capítulo de fundamentação teórica aborda os aspectos da que devem ser considerados na navegabilidade robótica, seguido da descrição de algumas das técnicas de aprimoramento da navegação que são de especial interesse para o

desenvolvimento deste trabalho. Ainda neste capítulo são descritas as funcionalidades e configurações presentes no *kit* LEGO Mindstorms e, por fim, são apresentados os trabalhos correlatos que serviram como base para o desenvolvimento das aplicações. O capítulo de desenvolvimento aborda as ferramentas, técnicas e trechos de códigos relevantes para o entendimento das funcionalidades implementadas. Por fim, são apresentados os resultados e as conclusões provenientes do desenvolvimento trabalho, além de possibilidades de futuras extensões.

2 FUNDAMENTAÇÃO TEÓRICA

Inicialmente, é apresentada a seção 2.1 que trata do conceito da navegação robótica e a dependência e influência do uso dos sensores para estimação de posição. Na seção 2.2 são abordados alguns dos problemas encontrados na navegação e a alternativa proveniente da união dos valores dos sensores. Na seção 2.3 é apresentada a estrutura do *kit* LEGO Mindstorms e a possibilidade de implementar projetos que integrem o robô e o computador. Por fim, a seção 2.4 apresenta os trabalhos correlatos.

2.1 NAVEGAÇÃO ROBÓTICA

A navegação robótica consiste na atividade em que o robô, a partir de uma posição inicial, deve atingir uma nova posição dentro de um determinado intervalo de tempo. Esta posição é composta pelos valores de x e y, que representam as coordenadas em um plano, e o valor de orientação Θ (PEDROSA, 2001, p. 9; PETROVIĆ; IVANJKO; PERIĆ, 2002, p. 91). De acordo com Borenstein, Everett e Feng (1996, p. 10) de maneira bastante generalista é possível dividir a tarefa da definição da posição robótica em duas vertentes: medida do posicionamento absoluto e relativo.

O posicionamento absoluto representa a medida do posicionamento global do robô em relação a um ponto fixo absoluto. Diferentemente do posicionamento relativo, não depende das medições e suas possíveis variações para definir o posicionamento. Necessita apenas das informações atualizadas do ambiente (NASCIMENTO, 2014, p. 8). *Active beacons* ou *Artificial Landmark Recognition* são dois métodos utilizados para estimar o posicionamento absoluto. No caso do método de *Active beacons* a posição é calculada a partir da incidência do sinal de três ou mais transmissores sobre um receptor. No método *Artificial Landmark Recognition* o posicionamento é estimado a partir da "visualização" por parte do robô de três ou mais pontos de referência no ambiente (BORENSTEIN; EVERETT; FENG; 1996, p. 10).

O posicionamento relativo é determinado através dos dados do estado anterior do robô comparado com a leitura corrente a fim de estimar sua posição atual. Este valor é relativo ao ponto inicial e não representa o posicionamento global do robô no ambiente (NASCIMENTO, 2014, p. 8). Nesta categoria podem ser mencionados técnicas de medição da odometria e da navegação inercial. Para calcular o valor da odometria é considerado o valor da rotação do motor e, em alguns casos, sua direção. Já na navegação inercial é realizada a união dos dados de rotação e aceleração para estimar o posicionamento do robô (BORENSTEIN; EVERETT; FENG; 1996, p. 10).

Independente da técnica de medição selecionada, existe a necessidade da percepção de características próprias do robô e do ambiente para que seja possível realizar a navegação. De acordo com Couto (2012, p. 11) e Russel e Norvig (2004, p. 872) esta percepção é essencial para que um robô seja autônomo. Através da percepção das características desejadas ele tem a possibilidade de tomar ações a partir de um sistema de controle inteligente. Ao finalizar esta ação é necessário que ele novamente atualize os valores referentes às características internas e do ambiente para que possa realizar novas ações. Este ciclo de atividades possui duas dependências: dos sensores para realizar a percepção e dos atuadores para que seja possível intervir no ambiente, conforme pode ser visualizado na Figura 1.

Figura 1- Ciclo de percepção e ação de um robô autônomo

Fonte: Couto (2012, p. 11).

2.1.1 Sensores na robótica

Os sensores possibilitam a interação entre o robô e o ambiente em que está inserido (RUSSEL; NORVIG, 2004, p. 872). Quando se trata de robôs móveis os sensores possuem o papel de sintetizadores da representação e da constante atualização dos acontecimentos do ambiente. Esta característica dá ao robô a possibilidade se ser autônomo (FACELI, 2001, p. 6).

Os sensores podem ser divididos em passivos e ativos. Os passivos, assim como as câmeras, não iniciam a interação com o ambiente, apenas recebem sinais que são emitidos pelas fontes que os cercam. No caso de sensores ativos, a exemplo do sonar, são emitidos sinais para o ambiente esperando que este seja retornado para o sensor. Por conta deste

comportamento, os sensores ativos enviam um maior número de informações ao sistema e consequentemente despendem mais execução (RUSSEL; NORVIG, 2004, p. 872).

Em outra classificação, de acordo com Couto (2012, p. 12), os sensores são divididos em exteroreceptivos, que registram características externas ao robô; e proprioceptivos, que apontam características internas. Similar à essa divisão, Russel e Norvig (2004, p. 872), defendem a existência de três grupos de sensores conforme a finalidade desejada:

a) sensores que medem a proximidade até determinado ponto: existem os telêmetros que são capazes de captar a distância de objetos próximos, um exemplo são os sonares. Para distâncias ainda menores existem os sensores táteis, que, ao exemplo do sensor de toque do *kit* LEGO Mindstorms, reage ao movimento de pressionar e soltar (LEGO GROUP, 2014). Para distâncias longas há o exemplo do *Global Positioning System* (GPS), que disponibiliza informações de posicionamento utilizando como referência os sinais pulsantes emitidos por satélites;
b) sensores de tratamento de imagens do ambiente: a câmera é considerada um sensor, pois a partir das imagens recebidas é possível processá-las e extrair informações do ambiente. A visão estereoscópica também pertence a esse grupo e possui grande importância na robótica por permitir a obtenção de informações de profundidade do ambiente;
c) sensores de leitura das propriedades internas do robô (sensores proprioceptivos): nesta categoria são utilizados *encoders* de eixos que registram valores da movimentação dos motores de maneira incremental permitindo o cálculo de odometria.

Embora tenha sido descrito que os sensores representem a interface entre o robô e o ambiente é importante salientar que eles não se comportam como dispositivos perfeitos (COUTO, 2012, p. 13). Conforme descrito por Wolf et al. (2009, p. 285), individualmente, os sensores disponibilizam apenas uma "visão de mundo", incompleta, imperfeita e com a possibilidade de conter erros. Estes erros possuem as seguintes causas: truncamento de valores, ocasionado pelo processo da digitalização do valor lido ou pela limitação de resolução máxima do sensor, ou ocorrência de imperfeições na detecção. O conhecimento dessas limitações deve ser considerado na escolha dos sensores adequados ao objetivo ou à tarefa que o robô deva realizar (COUTO, 2012, p. 12-13).

Um segundo problema no uso dos sensores é em relação ao conteúdo das informações extraídas. Isto porque é bastante comum na robótica móvel a ocorrência de *sensor aliasing* em

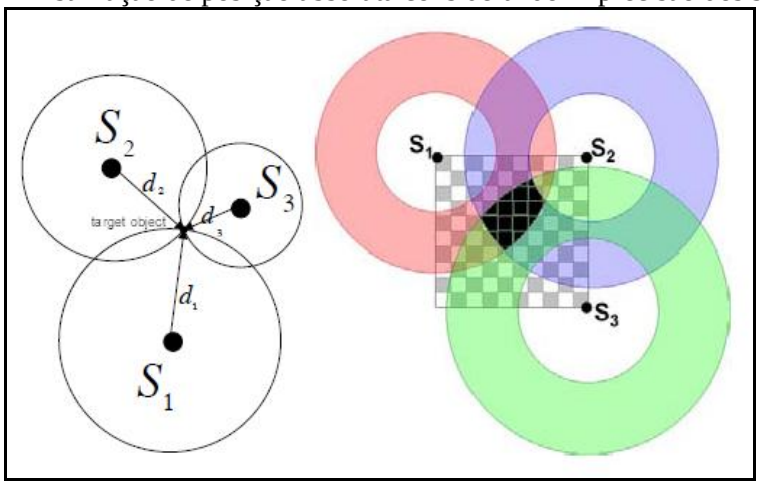
que os sensores robóticos registram os mesmos valores para lugares diferentes. Por exemplo, considerando a leitura da distância de um determinado objeto através de um sonar. Este sensor não fornece nenhuma informação adicional à respeito da composição ou cor do objeto, apenas a distância. Ainda que esta medição fosse realizada através de vários sonares, poderiam existir diversos pontos do ambiente que resultariam no mesmo conjunto de valores de medição (SUEGWART; NOURBAKHSH, 2004, p. 184).

2.1.2 Influência dos sensores na execução do posicionamento relativo e absoluto

Conforme já mencionado, a realização da navegação depende fundamentalmente da estimação da posição e orientação no ambiente. De acordo com Shareef et al. (2007, p. 323), no caso da estimação da posição absoluta de um robô autônomo, é possível utilizar uma rede de sensores. Estes sensores são interligados e dispostos em três pontos conhecidos do ambiente. A partir da distância que cada sensor estima em relação ao robô móvel autônomo é possível definir seu posicionamento representado pelo ponto de intersecção das três medições.

No entanto, esta técnica nem sempre representa o valor real do posicionamento por conta da imprecisão dos sensores. Na Figura 2 os sensores estão dispostos nos pontos conhecidos S1, S2 e S3. De cada um desses pontos é demonstrada a distância até o objeto, no caso um robô móvel autônomo, através dos valores de d1, d2 e d3. Estes valores são utilizados para estimar o ponto de intersecção que representa o valor da posição do robô no ambiente. Porém, em consequência da imprecisão dos sensores em detectar a distância até o robô, o valor de intersecção pode ser estimado em qualquer ponto da região sombreada (SHAREEF et al, 2007, p. 323).

Figura 2 – Estimação de posição absoluta considerando imprecisão dos sensores



Fonte: Shareef et al. (2007, p. 323).

Além do problema da imprecisão de sensores na estimação da posição absoluta, também deve ser considerada a necessidade de realizar constantemente a manutenção dos

transmissores. Além disso, quando há mudança na configuração do ambiente ou dos sensores é necessário manter atualizado o mapeamento e nem sempre as novas medidas são precisas (BORENSTEIN; FENG, 1994, p. 1).

Por outro lado, o posicionamento relativo é uma solução simples e barata de manter. O cálculo do deslocamento do robô autônomo é baseado no acompanhamento das rotações dos motores ligados às rodas, a partir de um ponto conhecido. A desvantagem desse cálculo, denominado odometria, é que são desconsiderados fatores físicos como deformidades do robô ou ocorrências de deslizamento que culminam no acúmulo ilimitado de erros. Como alternativa, há a possibilidade de melhorar a odometria através do tratamento das imprecisões das medições. Esta opção possibilita manter um sistema de robôs autônomos móveis a um custo reduzido já que a tarefa de estimar a posição é simplificada (PETROVIĆ; IVANJKO; PERIĆ, 2002, p. 91; BORENSTEIN; FENG, 1994, p. 1).

2.2 ODOMETRIA MELHORADA ATRAVÉS DO APRIMORAMENTO DA TÉCNICA DE NAVEGAÇÃO ROBÓTICA DEAD RECKONING (DR)

Originalmente, *Dead Reckoning* (DR) é um método matemático simples para determinar a posição atual de um navio (BORENSTEIN; EVERETT; FENG, 1996, p. 13). Para tanto, o cálculo faz uso da velocidade e curso anterior da embarcação para determinar a velocidade e o curso na futura posição. Esta nova posição estimada representa um valor aproximado, isto porque não são considerados problemas como o erro na leitura da bússola, erros do timoneiro ou mesmo os efeitos da corrente marítima. O navegador deve consultar a posição estimada quando necessário e utilizá-la para analisar as forças externas que agem na embarcação. A partir das posições estimadas e da análise de fatores de interferência, é possível realizar as correções necessárias para que o barco permaneça na rota desejada (BOWDITCH, 2002, p. 99).

Baseado na técnica náutica, os sistemas robóticos autônomos utilizam dos mesmos preceitos para realizar a navegação. Desta forma, a técnica de DR tem o papel de determinar a posição momentânea de um robô móvel autônomo. Através do valor do posicionamento é possível reparar os erros acumulados com a navegação através de correções recorrentes na rota realizada pelo robô autônomo (BORENSTEIN; EVERETT; FENG, 1996, p. 13).

Uma implementação comum da técnica de DR é a odometria. A odometria representa o valor de deslocamento do veículo de uma posição inicial conhecida até o ponto final que, utilizado em fórmulas matemáticas, possibilita estimar a posição robótica alcançada. Neste processo do cálculo de odometria, o valor de deslocamento comumente é extraído através de

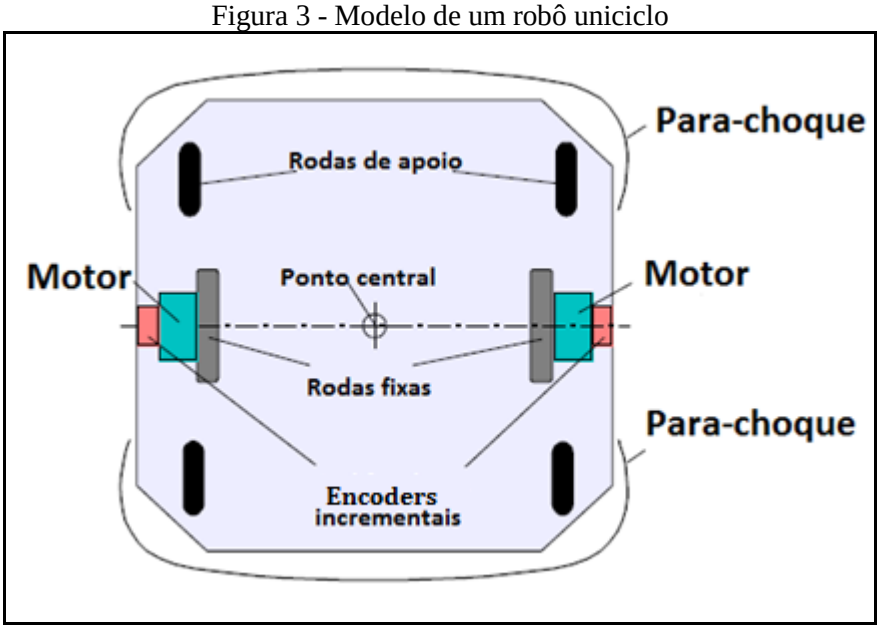
encoders incrementais acoplados aos motores que rotacionam as rodas do robô (PETROVIĆ; IVANJKO; PERIĆ, 2002, p. 91).

2.2.1 Cálculo de odometria

O cálculo de odometria é realizado enquanto o robô autônomo se movimenta. O valor de deslocamento é enviado continuamente ao processador central que atualiza o posicionamento do robô (x, y e Θ) de acordo com equações geométricas (BORENSTEIN; EVERETT; FENG, 1996, p. 13; PETROVIĆ; IVANJKO; PERIĆ, 2002, p. 91).

A forma de calcular a odometria está diretamente relacionada ao modelo do robô utilizado. Em geral, o robô uniciclo, também conhecido como robô móvel diferencial, é selecionado pelos pesquisadores para realização de testes que envolvem estratégias de controle de navegação. Este robô é composto por duas rodas fixas, que podem ser controladas de maneira independente. Cada roda é ligada a um motor que possui um *encoder* incremental, conforme pode ser visualizado na Figura 3 (SECCHI, 2008, p. 22; BORENSTEIN; FENG, 1994, p. 3).

Figura 3 - Modelo de um robô uniciclo



Fonte: baseado em Borenstein e Feng (1994, p. 3).

De acordo com Petrović, Ivanjko e Perić (2002, p. 92) o cálculo odométrico para a estimação da posição de um robô uniciclo é baseado na velocidade extraída dos *encoders* de ambas as rodas. A partir dos valores de velocidade é possível estimar as coordenadas x, y e a orientação Θ de um robô móvel autônomo no instante seguinte ao que o robô se encontra. A seguir estão as fórmulas que representam o cálculo de odometria de acordo com a cinemática do robô uniciclo:

a) $x_{k+1} = x_k + v_{totk} . T . \cos \Theta_{k+1}$;

b) $y_{k+1} = y_k + v_{totk} . T . \text{sen } \Theta_{k+1}$;

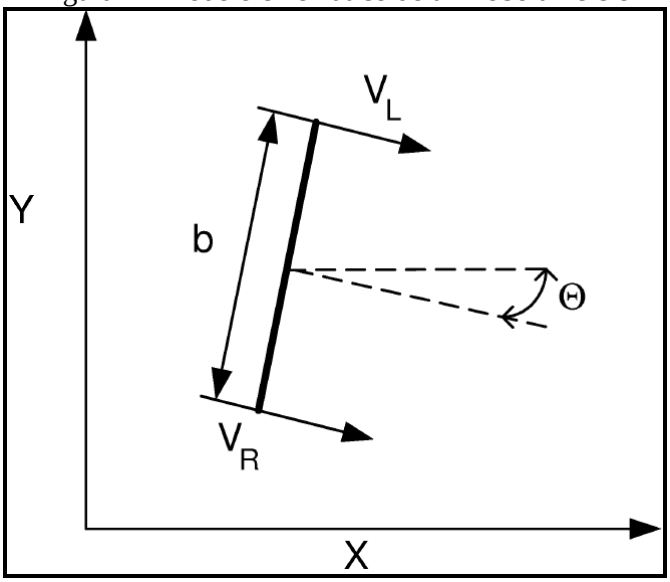
c) $\Theta_{k+1} = \Theta_k + \Delta\Theta_k$;

d) $v_{totk} = (vL_k + vR_k)/2$;

e) $\Delta\Theta_k = (180/\pi) . (vR_k - vL_k)/b . T$.

Os valores de x_k e y_k representam o ponto central do robô conforme demonstrado na Figura 3. T indica o intervalo de tempo de execução. v_{totk} designa a velocidade total de translação do robô (m/s). Θ_k simboliza o valor do ângulo entre o robô e o eixo x. vL_k e vR_k representam as velocidades dos *encoders* esquerdo e direito, respectivamente (m/s). Por fim, b indica a distância entre os eixos do robô (PETROVIĆ; IVANJKO; PERIĆ, 2002, p. 92). Para melhor compreender a função de cada variável no cálculo do posicionamento do robô autônomo, a Figura 4 demonstra o modelo cinemático de um robô móvel autônomo uniciclo.

Figura 4 - Modelo cinemático de um robô uniciclo



Fonte: Petrović, Ivanjko e Perić (2002, p. 92).

Conforme demonstrado a partir do cálculo de DR é possível verificar que as fórmulas são baseadas nos valores de velocidade dos *encoders* para incrementar as posições de x, y e Θ. Portanto, esta técnica assume que não ocorrem erros nas medições dos *encoders*. Esta é uma visão limitada da realidade e representa a desvantagem desta implementação. Pois ao desconsiderar as possíveis falhas durante a navegação, possibilita o acumulo ilimitado de erros (BORENSTEIN; EVERETT; FENG, 1996, p. 13; PETROVIĆ; IVANJKO; PERIĆ, 2002, p. 91; BORENSTEIN; FENG, 1994, p. 4).

Os erros encontrados na estimação da posição de um robô através da odometria podem ser classificados em sistemáticos e não sistemáticos. Sistemáticos são os erros inerentes à

composição do robô, por exemplo: diâmetros das rodas desiguais e desalinhamento de rodas. Não sistemáticos são os erros não previstos, por exemplo: existência de objetos inesperados no caminho e derrapagens devido ao piso escorregadio, piso irregular e giros rápidos. Entre essas duas categorias, o sistemático é o mais grave, já que os erros são acumulados constantemente. Para que esses erros sejam tratados é importante detectá-los (BORENSTEIN; EVERETT; FENG, 1996, p. 130; PETROVIĆ; IVANJKO; PERIĆ, 2002, p. 91). Uma das formas de perceber os erros de naveção robótica é através do uso de métodos de FS.

2.2.2 Fusão de Sensores (FS)

O conceito de fusão de informações não é novo. Isto se deve ao fato de que os seres humanos e animais fazem uso de seus sentidos em auxílio da própria sobrevivência. Por exemplo, ao verificar as características de uma substância comestível seria extremamente ineficaz se fosse utilizado apenas o sentido da visão. Portanto, a fusão de dados multissensoriais é importante para que as informações recebidas sejam mais precisas (LIGGINS; HALL; LLINAS, 2009, p. 1).

Embora o conceito seja antigo, a literatura relacionada à FS é relativamente nova datando da década de 1980 (SALUSTIANO, 2006, p. 6). Isto se deve à dependência do avanço em software e hardware que garantem a criação de sensores que emulam capacidades humanas (LIGGINS; HALL; LLINAS, 2009, p. 1).

Em relação à nomenclatura, ainda existem divergências quanto ao termo FS. Dentre as existentes, são exemplos: Fusão de Dados, Fusão de Sensores (FS), Fusão de Múltiplos Sensores e Fusão de Informação (SALUSTIANO, 2006, p. 6). Neste trabalho será tratada especificamente a FS, já que os dados são originados de sensores artificiais que não são especificamente do mesmo tipo. Esta é distinta, por exemplo, da Fusão de Dados, que trata da união de informações provenientes de sensores, humanos, relatórios, bancos de dados, entre outras fontes (LIGGINS; HALL; LLINAS, 2009, p. 1).

De maneira geral, os métodos de FS combinam a informação de vários sensores a fim de realizar inferências mais específicas se comparado a utilização de um sensor independente (LIGGINS; HALL; LLINAS, 2009, p. 1). Através da FS é possível unir dados resultando na obtenção de informações que são superiores a capacidade isolada de cada sensor, especialmente em relação à confiabilidade e precisão.

Através da FS o sistema passa a ser tolerante a falhas já que não depende de apenas uma origem de dados. Outra vantagem no uso da FS é a possibilidade de utilizar vários sensores simples e baratos no lugar de um que seja de valor elevado ou de funcionamento

complexo (FACELI, 2001, p. 22). No próximo capítulo são abordados o método de FS Filtro de Kalman e o método de FS Rede Neural de Função Radial para correção da imprecisão de odometria.

2.2.3 Estimação de posicionamento robótico através do método do método de FS filtro de Kalman

O filtro de Kalman é utilizado quando é necessário avaliar o estado de um fenômeno físico a partir de visualizações com medições imprecisas ao longo do tempo (RUSSEL; NORVIG, 2004, p. 536). Aplicações práticas deste método compreendem o acompanhamento acústico de submarinos, o acompanhamento visual de veículos e pessoas, a reconstrução de trajetórias de correntes oceânicas, entre outras aplicações. Dentre as mencionadas, a mais clássica é o acompanhamento por radar de aeronaves e mísseis (RUSSEL; NORVIG, 2004, p. 541).

O uso do filtro de Kalman para implementação da FS é indicado para quando a leitura dos múltiplos sensores é direcionada a um mesmo fenômeno físico. Neste caso os dados coletados do sensor podem ser combinados de forma direta (LIGGINS; HALL; LLINAS, 2009, p. 8). No caso específico da estimação de posicionamento robótico, o filtro de Kalman pode ser utilizado para combinar os valores de odometria e orientação proporcionando um aumento significativo na acurácia da navegação robótica autônoma (PETROVIĆ; IVANJKO; PERIĆ, 2002, p. 92).

A desvantagem da utilização deste método reside no fato de que ele exige a utilização constante dos sensores que participam do processo de FS durante a realização da navegação robótica. Como alternativa, é possível utilizar uma Rede Neural Artificial (RNA) que exige o uso dos sensores apenas na fase de treinamento. Através dos valores extraídos dos sensores, é possível “ensinar” à rede qual é o comportamento de navegação do robô de forma a inferir seu posicionamento ao longo do trajeto (PETROVIĆ; IVANJKO; PERIĆ, 2002, p. 92).

2.2.4 Correção de erros de navegação sistemáticos através do método de FS Rede Neural de Função Radial (RNFR)

A Rede Neural de Função Radial (RNFR), assim como as demais RNAs, é caracterizada pela capacidade de aprendizagem. O conhecimento é adquirido através de um conjunto de padrões ou exemplos que são repassados à RNA. A partir destas entradas o algoritmo sintetiza as características que representam a informação recebida podendo, inclusive, gerar respostas a novos padrões (FACELI, 2001, p. 43).

As RNAs são compostas por nós e em cada um deles é associado determinado peso. Esta é a principal forma de armazenar informação durante a execução do modelo. A aprendizagem ocorre porque parte dos nós é ligado ao mundo externo, portanto recebem informações do ambiente. Conforme são recebidos esses valores nos nós de entrada os pesos são ajustados. De maneira geral, os nós possuem processamento próprio e não dependem de supervisão (RUSSELL; NORVIG, 2004, p. 567);

As RNAs podem sem utilizadas como um método de FS em auxílio à navegação robótica porque possuem a capacidade de extrair características e estabelecer padrões do ambiente a partir de um vetor composto de vários valores de sensores (LIGGINS; HALL; LLINAS, 2009, p. 8). Estas características são extraídas porque as RNA, em especial a RNFR, têm a capacidade de estabelecer uma relação não linear entre características como velocidade dos *encoders* do robô e o valor de sua orientação (PETROVIĆ; IVANJKO; PERIĆ, 2002, p. 93).

E relação à estrutura da RNFR, ela possui geralmente uma camada intermediária (não linear) associada a funções de base radial. Comumente, é selecionada a função Gaussiana. O treinamento costuma ser dividido em duas fases, sendo elas: a não-supervisionada e a supervisionada. Na fase não-supervisionada a rede é submetida à uma rotina que posiciona os seus centros em regiões do espaço que se encontram os vetores de entrada de maior representatividade. Na segunda fase, os pesos da camada de saída são ajustados de forma a transformar os valores provenientes da camada intermediária em saídas lineares (FACELI, 2001, p. 46).

2.3 LEGO MINDSTORMS

O LEGO Mindstorms foi criado no ano de 1998 passando pelas versões RCX do mesmo ano, NXT e a atual EV3 lançada no ano de 2013, com componentes que possibilitam a criação de robôs. O presente trabalho utilizará a versão NXT 2.0 que é composta por 612 peças que compreendem um bloco programável, sensores, motores, cabos conversores e peças de montagem (LEGO GROUP, 2014).

Por padrão, o kit contém quatro tipos de sensores: um sensor ultrassônico, capaz de calcular a distância até um objeto e responder a movimentos; dois sensores de toque que registram o movimento de pressionar e soltar; e o sensor de luz que capta variações de luz e cor. No entanto, é possível acoplar outros sensores como a bússola que auxilia na acurácia da navegação do robô (LEGO GROUP, 2014).

O bloco programável, denominado NXT, pode ser conectado a motores e sensores que permitem as ações do agente Mindstorms. Este bloco possui as seguintes especificações técnicas: um microcontrolador de 32-bits ARM7 que possui acesso a 64 KB de RAM e 256 KB de memória flash, uma tela *Liquid Crystal Display* (LCD) com exibição gráfica de 100 x 64 pixels, 4 entradas para conexão dos sensores e 3 portas de saída e um alto-falante com qualidade de som a 8KHz. Esse bloco pode ter como fonte de alimentação 6 pilhas AA ou uma bateria recarregável. A partir de uma entrada USB com velocidade de 12 Mbit/s é possível conectar o bloco programável ao computador (LEGO GROUP, 2014).

Junto com o kit de montagem, é disponibilizado ao usuário o software que, de maneira intuitiva, auxilia iniciantes a realizar a programação do robô. Contudo, existem alternativas de implementação de *Application Programming Interface* (API) para programação do NXT. A biblioteca a ser utilizada nesse trabalho é a LeJOS NXJ que, por ser uma máquina virtual Java, possui as seguintes características: linguagem orientada a objetos, possibilidade de implementações utilizando *threads*, existência de matrizes multidimensionais, recursão, sincronização e tratamento de exceções. Através desta biblioteca é possível implementar aplicativos para serem executados no robô com acesso às propriedades dos sensores e motores. Por conta das limitações de processamento e armazenamento do robô, também é disponibilizada uma API para implementação de aplicativos no computador integrados a aplicativos executados no bloco programável (LEJOS, 2014).

2.4 TRABALHOS CORRELATOS

O primeiro trabalho correlato é bastante similar ao objetivo deste trabalho já que trabalha com RNFR e cálculo de odometria na estimação da posição robótica. O segundo trabalho correlato traz exemplos de técnicas de estimação de posição absoluta, para tanto utiliza um robô baseado na plataforma LEGO Mindstorms. O último trabalho aborda, entre outros métodos, a teoria, implementação e aplicabilidade da RNFR.

2.4.1 *Neural Network Based Correction of Odometry Errors in Mobile Robots*

Este trabalho aborda a comparação de três métodos de estimação da posição robótica: cálculo de odometria, RNFR e Filtro de Kalman. Os autores Petrović, Ivanjko e Perić (2002, p. 91) alegam que o Filtro de Kalman representa o método de maior precisão na estimação da posição robótica. No entanto, este método exige que o robô disponha de sensores adicionais como a bússola, por exemplo.

Como alternativa eles propõe uma estrutura de RNFR que tem o propósito de calcular a variação odométrica de um robô e auxiliar o cálculo de odometria. O treino dessa RNA é é realizado utilizando como base os valores de orientação e velocidade calculados pelo Filtro de Kalman (PETROVIĆ; IVANJKO; PERIĆ, 2002, p. 93).

O resultado desse trabalho apresenta a diferença entre a posição estimada e a posição real do robô no ambiente para os três métodos implementados. Nessa comparação, o Filtro de Kalman obteve melhores resultados, seguido dos valores obtidos pela RNFR.

2.4.2 Multi Sensor Fusion Framework for Indoor-Outdoor Localization of Limited Resource Mobile Robots

Marín et al (2013, p. 12133) demonstra o uso da FS para auxílio da estimação de posição de robôs móveis que possuem recursos computacionais limitados. A implementação usa como base a plataforma LEGO Mindstorms NXT.

Para verificar o posicionamento do robô dentro do ambiente é necessário levar em consideração, por exemplo, dados como aceleração, velocidade e rotação. No caso, são utilizados um sensor global (câmera) e uma unidade de medição inercial. A programação é baseada em eventos e garante que sempre que houver um desvio de rota do robô além do estipulado, o sensor global deve ser acionado para auxiliar na correção da rota (MARÍN et al., 2013, p. 14134).

Como os sensores são sujeitos a ruídos, além de possuírem diferentes níveis de precisão, é utilizado o filtro de Kalman na implementação da FS. São realizados diversos testes que evidenciam, dentre outras características, a eficiência da utilização do método implementado para estimação da posição de robôs em ambientes internos e externos. Além disto, verifica-se a vantagem de economia de processamento para o robô, já que o sensor global é acionado apenas quando necessário (MARÍN et al., 2013, p. 14155).

2.4.3 Combinação de métodos de Inteligência Artificial para fusão de sensores

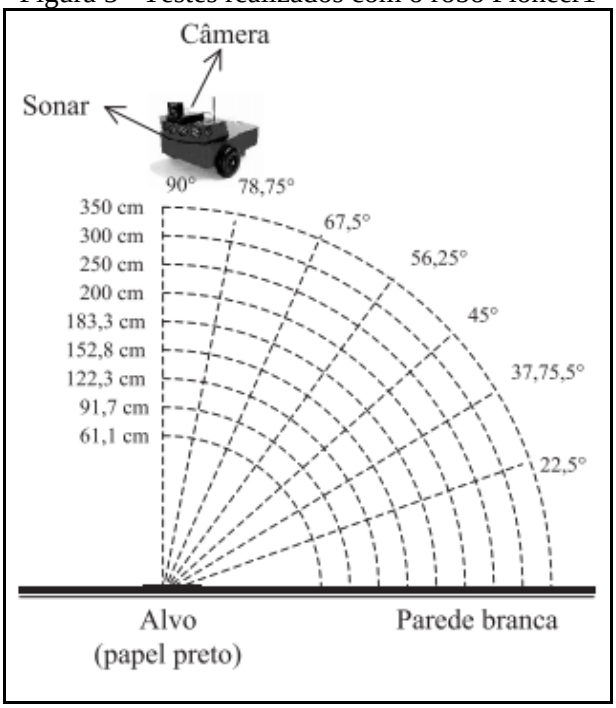
Faceli (2001, p. 42) investiga o uso de diferentes métodos de Inteligência Artificial (IA) utilizados na implementação da FS. O propósito é melhorar a precisão e acurácia dos dados obtidos por esses dispositivos individualmente. Para tanto, são utilizados diversos algoritmos, de diferentes paradigmas (conexionista, simbólico e estatístico), para implementação de FS, por considerar que cada um possui características próprias e por vezes complementares.

No trabalho foram implementados os algoritmos de RNAs do tipo *MultiLayer Perceptron* (MLP), *Radial Basis Function* (RBF) e *Cascade Correlation*. Além destes também foram implementados o *Support Vector Machine* (SVM), *Cubist* e M5. Para cada um destes algoritmos foi determinada a melhor combinação de implementação e sensores. Estas combinações foram denominadas de estimadores. Conforme Faceli (2001, p. 51), como os estimadores resultam em valores diferenciados, além de apenas utilizá-los isoladamente é explorado o uso destes em grupo, em diferentes combinações, denominados comitês.

O cenário abordado no trabalho trata da FS para melhoria da estimação da distância até um ponto estimado por dois modelos de robôs: Pioneer1 (com uma câmera e sete sonares) e B14 (oito sonares, oito sonares visuais e oito sonares de infravermelho). Os dados extraídos e utilizados para definição dos estimadores contam com as medições obtidas de cada sensor dos robôs em diferentes distâncias e ângulos. A Figura 5 demonstra os cenários aplicados ao robô Pionner1.

Como resultado, foi possível obter a redução de erro do melhor sensor em até 89,35% no robô Pioneer1 (FACELI, 2001, p. 106) e de 97,77% do melhor sensor do robô B14 (FACELI, 2001, p. 120). Estes dados demonstram a eficácia do uso de métodos de IA para a implementação de FS pois possibilitam a melhora da precisão e acurácia na estimação de sensores individuais (FACELI, 2001, p. 124).

Figura 5 - Testes realizados com o robô Pioneer1



Fonte: Faceli (2001, p. 62).

3 DESENVOLVIMENTO

Neste capítulo serão apresentadas as etapas de desenvolvimento do trabalho. No capítulo 3 serão descritos os requisitos que embasam o desenvolvimento do software. A seguir, é demonstrada a especificação e a implementação através de trechos de código e exemplos de uso do software. Por fim, são descritos os resultados e as discussões pertinentes ao trabalho realizado.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

O software proposto deverá:

a) criar um robô a partir do kit LEGO Mindstorms provido de dois sensores de odometria (requisito não-funcional – RNF);
b) implementar o envio dos valores de odometria do robô para o computador via *bluetooth* (requisito funcional – RF);
c) implementar a leitura dos valores de posicionamento enviados do computador para o robô via *bluetooth* (RF);
d) implementar o envio do valor da bússola de um celular para o computador via *bluetooth* (RF);
e) implementar a leitura dos valores de odometria e bússola enviados ao computador via *bluetooth* (RF);
f) implementar a navegação robótica a partir do método de RNFR (RF);
g) implementar a navegação robótica baseada no cálculo de odometria (RF);
h) implementar a exibição visual da rota estimada a partir do cálculo de odometria no protótipo (RF);
i) implementar a exibição visual da rota estimada a partir do método de RNFR no protótipo (RF);
j) permitir comparar a posição final estimada através do cálculo de odometria e do método RNFR, com a posição real do robô no ambiente (RF);
k) utilizar a API LeJOS para implementação da regra de execução do robô (RNF);
l) implementar no ambiente de desenvolvimento Eclipse (RNF);
m) implementar na linguagem Java (RNF).

3.2 ESPECIFICAÇÃO

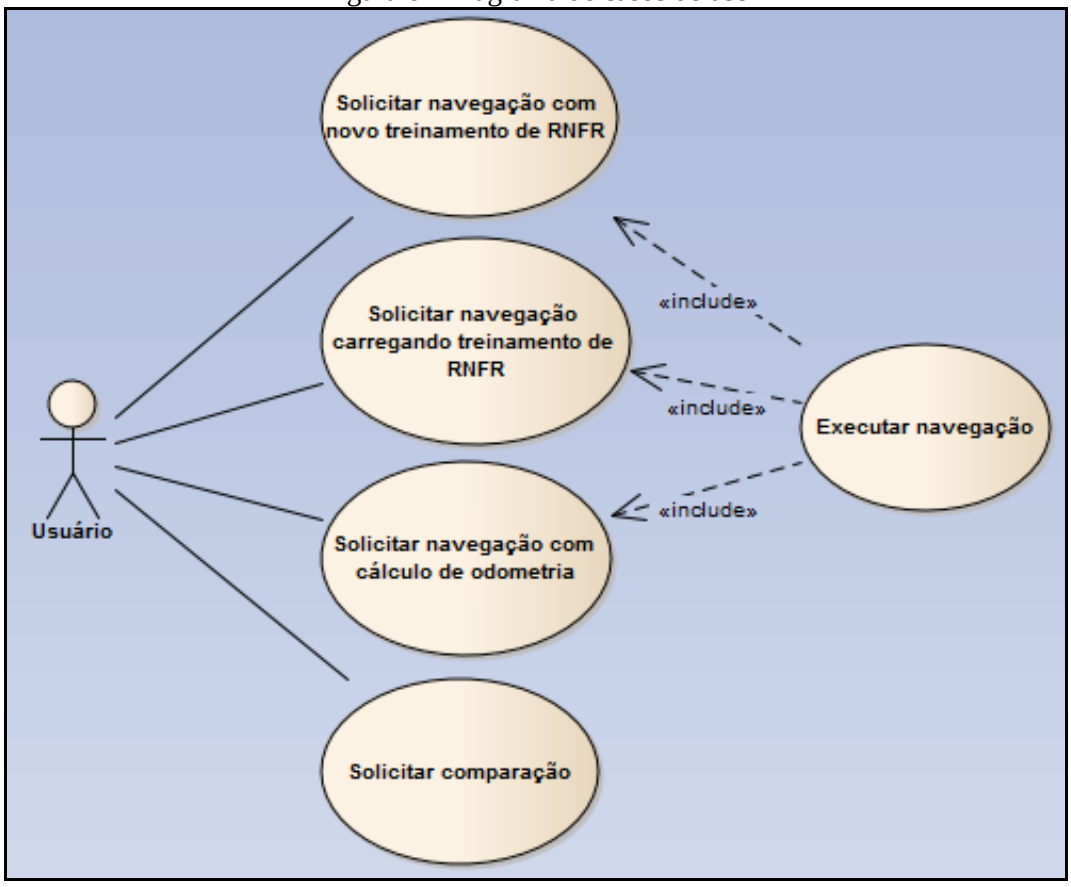
As especificações das implementações foram criadas utilizando *Unified Modeling Language* (UML) por intermédio da ferramenta *Enterprise Architect* (EA). Nos capítulos

seguintes são apresentados o diagrama de caso de uso, classes e atividades das funcionalidades desenvolvidas.

3.2.1 Diagrama de casos de uso

O software implementado possui cinco casos de uso conforme Figura 6: `Solicitar navegação com novo treinamento de RNFR`, `Solicitar navegação carregando treinamento de RNFR`, `Solicitar navegação com cálculo de odometria`, `Solicitar comparação` e `Executar navegação`. A seguir estes casos de uso serão detalhados respectivamente. No caso de uso `UC01` o `Usuário` do software solicita a realização do treino da RNFR para então execução da navegação robótica baseada neste método de FS. No `UC02` o `Usuário` possui a alternativa de solicitar a navegação robótica através do método de RNFR, sem a necessidade de criar um treinamento para a RNA. No `UC03` o `Usuário` pode solicitar a execução da navegação robótica baseada no cálculo de odometria. O `UC04` representa a execução da navegação robótica. Por fim, o `Usuário` pode comparar os dados das navegações robóticas solicitadas, conforme `UC05`.

Figura 6 - Diagrama de casos de uso



O caso de uso UC01 contém as ações que o Usuário deve executar para que seja possível solicitar o treino da RNFR e a execução da navegação através deste método de FS. Mais detalhes a respeito do UC01 podem ser verificados no Quadro 1.

Quadro 1 - UC01 – Solicitar navegação com novo treinamento de RNFR

UC01 - Solicitar navegação com novo treinamento de RNFR	
Descrição	O objetivo deste caso de uso é criar um novo arquivo contendo os dados de treinamento da RNFR. Utilizar estes dados para iniciar a RNFR possibilitando a execução da navegação baseada nesse método de FS.
Ator	Usuário
Pré-condições	O protótipo deve estar executando. O robô deve estar ligado e com a opção de receber conexões via *bluetooth* ativada. O robô deve estar executando o aplicativo que lê e envia os valores de odometria para o computador. O celular deve estar ligado e com a opção de receber conexões via *bluetooth* ativada. O celular deve estar executando o aplicativo que envia os valores da bússola ao computador.
Cenário Principal	01) O usuário deve selecionar no campo Cenários o modelo de cenário que o robô deve executar a navegação. 02) O usuário deve selecionar o método RNFR no campo Método; 03) O usuário deve clicar no botão Executar. 04) O protótipo pergunta se o usuário deseja criar um novo treinamento. 05) O usuário clica no botão Sim. 06) É criado um novo arquivo com a extensão .trn. 07) O robô executa os passos do treinamento e os dados extraídos são gravados no arquivo .trn. 08) A partir dos dados treinados o robô inicia a navegação conforme o cenário selecionado no passo 01. 09) Os dados de navegação são exibidos no protótipo. 10) O usuário deve indicar no campo Posição Real os valores de x e y do robô no ambiente. 11) Os dados são gravados em arquivo com a extensão .rnfr.

O UC02 é similar ao UC01, no entanto não é realizado o treinamento da RNFR. Conforme será explicado no capítulo 3, que trata das técnicas de desenvolvimento utilizadas, a execução da navegação robótica a partir do método de RNFR não necessita dos valores da bússola. A descrição detalhada deste caso de uso pode ser conferida no Quadro 2.

Quadro 2 - UC02 – Solicitar navegação carregando treinamento de RNFR

UC02 - Solicitar navegação carregando treinamento de RNFR	
Descrição	O objetivo deste caso de uso é a solicitação da navegação robótica via método de RNFR.
Ator	Usuário
Pré-condições	O protótipo deve estar executando. O robô deve estar ligado e com a opção de receber conexões via *bluetooth* ativada. O robô deve estar executando o aplicativo que lê e envia os valores de odometria para o computador.
Cenário Principal	01) O usuário deve selecionar no campo Cenários o modelo que o robô deve executar a navegação. 02) O usuário deve selecionar o método RNFR no campo Método; 03) O usuário deve clicar no botão Executar. 04) O protótipo pergunta se o usuário deseja criar um novo treinamento. 05) O usuário clica no botão Não. 06) É exibida uma tela com os arquivos com a extensão .trn 07) O usuário seleciona um desses arquivos. 08) A partir dos dados treinados o robô inicia a navegação conforme o cenário selecionado no passo 01. 09) Os dados de navegação são exibidos no protótipo. 10) O usuário deve indicar no campo Posição Real os valores de x e y do robô no ambiente. 11) Os dados são gravados em arquivo com a extensão .rnfr.

O caso de uso UC03 contém as ações que o Usuário deve executar para que seja possível solicitar a execução da navegação através do cálculo de odometria (mais detalhes do cálculo no capítulo 2.2). O detalhamento do UC03 pode ser verificado no Quadro 3.

Quadro 3 - UC03 – Solicitar navegação com cálculo de odometria

UC03 – Solicitar navegação com cálculo de odometria	
Descrição	O objetivo deste caso de uso é a solicitação da navegação robótica via cálculo de odometria.
Ator	Usuário
Pré-condições	O protótipo deve estar executando. O robô deve estar ligado e com a opção de receber conexões via *bluetooth* ativada. O robô deve estar executando o aplicativo que lê e envia os valores de odometria para o computador.
Cenário Principal	01) O usuário deve selecionar no campo Cenários o modelo que o robô deve executar a navegação. 02) O usuário deve selecionar o método Cálculo de Odometria no campo Método; 03) O usuário deve clicar no botão Executar. 04) O robô inicia a navegação conforme o cenário selecionado no passo 01. 05) Os dados de navegação são exibidos no protótipo. 06) O usuário deve indicar no campo Posição Real os valores de x e y do robô no ambiente. 07) Os dados são gravados em arquivo com a extensão .co.

O caso de uso UC04 é executado a partir da solicitação de navegação via método de RNFR descrita pelo caso de uso UC01 e UC02, ou solicitação de navegação via cálculo de odometria, detalhada através do UC03. Seu detalhamento é realizado no Quadro 4.

Quadro 4 - UC04 – Executar navegação

UC04 – Executar navegação	
Descrição	O objetivo desse caso de uso é executar a navegação robótica de acordo com o modelo de cenário e o método de execução de navegação selecionado (RNFR ou cálculo de odometria) pelo usuário.
Ator	Usuário
Pré-condições	Computador e robô conectados via *bluetooth*. O robô deve estar executando o aplicativo que lê e envia os valores de odometria para o computador.
Cenário Principal	1) O robô envia ao protótipo os valores de velocidade dos *encoders* acoplados à roda esquerda e direita. 2) O protótipo recebe os valores de velocidade enviados pelo robô. 3) O protótipo calcula a posição do robô de acordo com o método de execução de navegação selecionado. 4) O protótipo verifica se a posição estimada está de acordo com o cenário selecionado. 5) O protótipo envia ao robô as velocidades dos *encoders* esquerdo e direito coerentes à execução do cenário selecionado. 6) O robô recebe os valores de velocidade do protótipo e os aplica aos *encoders* do robô

Após a execução da navegação via RNFR e cálculo de odometria, o Usuário tem a possibilidade de comparar a posição estimada por cada método com a posição real do robô no ambiente. O detalhamento dessas ações pode ser verificado no caso de uso UC05 de acordo com o Quadro 5.

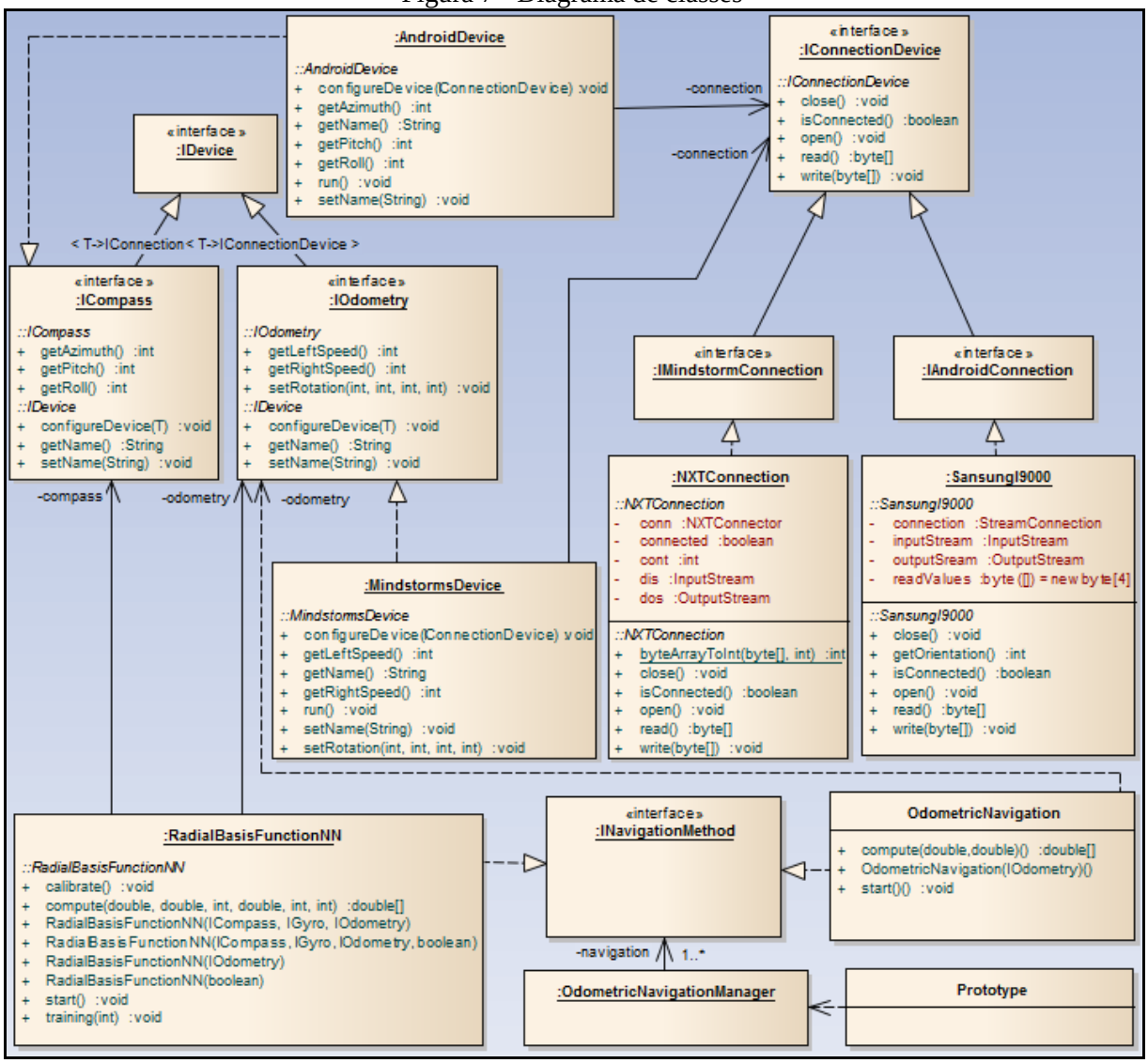
Quadro 5 - UC05 – Solicitar comparação

UC05 – Solicitar comparação	
Descrição	O objetivo desse caso de uso é fazer uso dos arquivos gerados a partir da execução dos métodos de navegação via RNFR e cálculo de odometria para exibir no protótipo a diferença entre a posição estimada e a posição real do robô no ambiente.
Ator	Usuário
Pré-condições	Ter, ao menos, um arquivo com a extensão .rnfr. Ter, ao menos, um arquivo com a extensão .co.
Cenário Principal	1) O usuário deve selecionar a aba Comparação do protótipo. 2) O usuário deve clicar no campo RNFR e selecionar um arquivo que contenha a extensão .rnfr. 3) O usuário deve clicar no campo Cálculo de Odometria e selecionar um arquivo que contenha a extensão .co. 4) O usuário deve clicar no botão Comparar. 5) O protótipo deve exibir os dados de navegação e a distância da posição final e real do robô para ambos os métodos.

3.2.2 Diagrama de classes

Na Figura 7 podem ser observadas as principais classes e interfaces que compõem o software implementado. Neste diagrama de classes foi optado por exibir os procedimentos e atributos de maior relevância no processo de execução da navegação robótica.

Figura 7 - Diagrama de classes



A classe `Prototype` representa a interface gráfica de exibição e comparação das rotas percorridas pelo robô autônomo de acordo com o método de execução da navegação robótica selecionado pelo usuário. `Prototype` possui uma associação para a classe `OdometricNavigationManager`, que representa a entrada para acesso às funcionalidades implementadas. Esta classe contém uma coleção dos possíveis métodos de execução de navegação robótica implementados. Cada um desses métodos é representado pela interface `INavigationMethod`. A partir desta interface e de acordo com o proposto pelo trabalho (capítulo 1.1), foram criadas as classes `RadialBasisFunctionNN` e `OdometricNavigation`,

que representam a implementação dos métodos RNFR e cálculo de odometria, respectivamente. Foi optado por criar a interface `INavigationMethod` para que seja possível adicionar outras implementações de métodos de execução em possíveis extensões do presente trabalho.

A classe `RadialBasisFunctionNN` e `OdometricNavigation` possuem o método `start` que inicializa o processo de navegação. Iniciada a navegação, é possível chamar o método `compute` que recebe os valores de velocidade dos *encoders* do robô e calcula a posição estimada devolvendo um `array` com `x`, `y` e `Θ` estimados. Para que esta navegação seja realizada, as classes `RadialBasisFunctionNN` e `OdometricNavigation` são ligadas aos dispositivos físicos. As propriedades desses dispositivos são representadas pela interface `IDevice`.

Os métodos RNFR e cálculo de odometria necessitam acessar os valores da bússola e da odometria do robô. Para tanto, foram criadas as interfaces `ICompass` e `IOdometer` que possuem as propriedades necessárias em um dispositivo para a execução dos métodos de navegação robótica. As implementações dessas interfaces foram desenvolvidas de acordo com os dispositivos físicos disponíveis. Portanto, para a interface `ICompass` foi criada a classe `AndroidDevice`, e para a classe `IOdometer` foi criada a classe `MindstormsDevice`. A função dessas classes é estabelecer e manter conexão com os dispositivos físicos.

A conexão com os dispositivos físicos é representada pela interface `IConnectionDevice`. Esta interface foi criada pois é possível ter a mesma configuração (uma extensão de `IDevice`) para diferentes dispositivos físicos compatíveis. Essas conexões são expressas a partir das classes `NXTConnection` (conexão com bloco programável NXT) e `SansungI9000` (conexão com o celular que possui a bússola).

3.2.3 Diagrama de atividades

Na Figura 8 é possível verificar o fluxo de atividades necessárias para que seja executada a navegação através dos métodos de RNFR e cálculo de odometria. Neste diagrama também estão os pré-requisitos para a realização da comparação entre esses métodos.

O fluxo é iniciado com a possibilidade do `Usuário` optar pela execução de um método de navegação ou pela comparação dos dados das execuções anteriormente realizadas. Caso o `Usuário` escolha a execução da navegação, ele deve optar entre o método RNFR ou o método do cálculo de odometria. A execução de ambos os métodos é disponibilizada através do

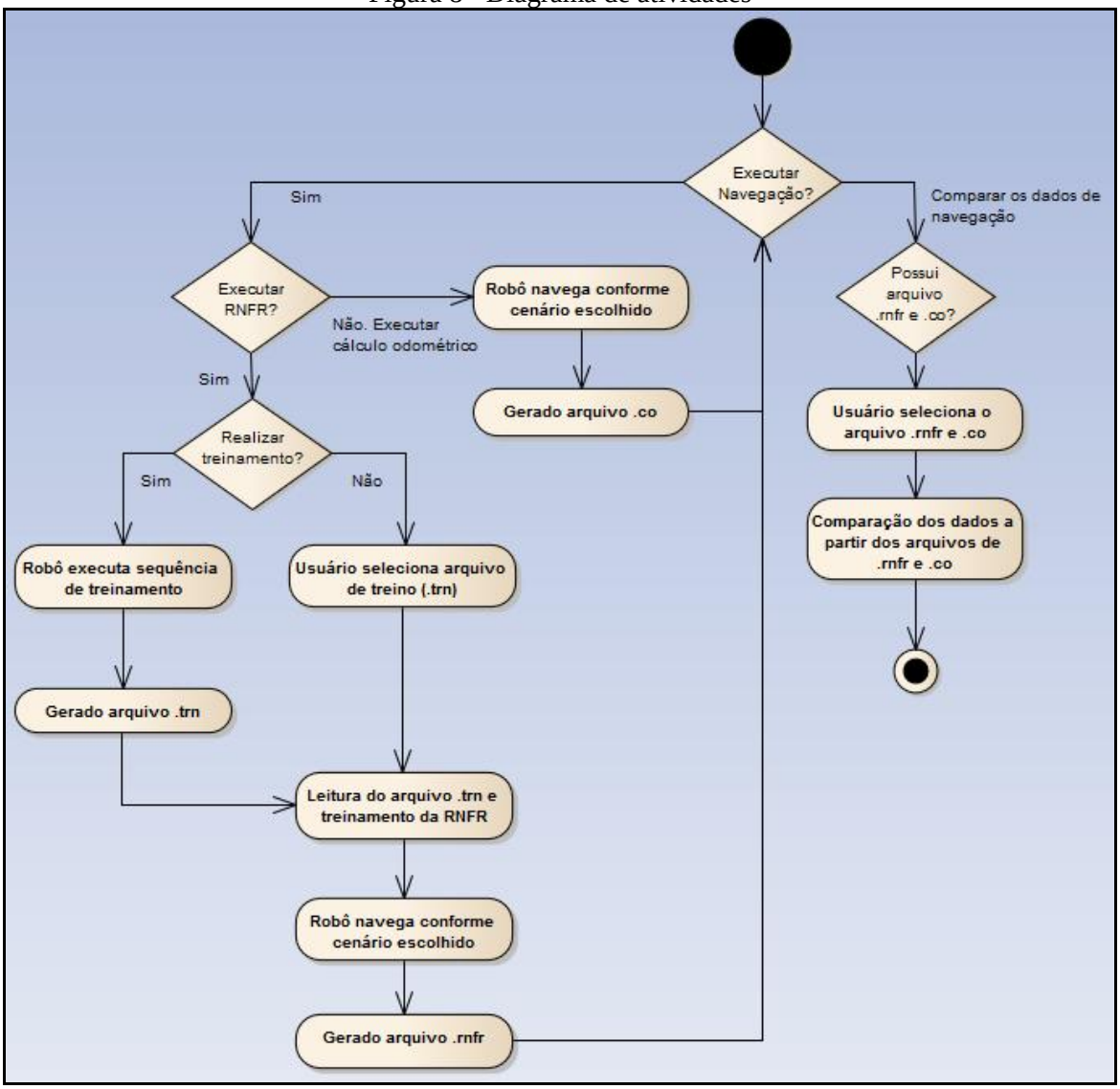
protótipo e em ambas as situações o Usuário deve selecionar o modelo de cenário que o robô vai executar.

Se o Usuário desejar executar o método RNFR, o protótipo vai questionar se deve ser realizado o treinamento. Caso o Usuário clique no botão Sim, o robô iniciará uma sequência de movimentos que são gravados em um arquivo .trn que posteriormente é utilizado para o treinamento da RNFR. Caso o Usuário clique no botão Não, deve ser selecionado um arquivo .trn existente que é utilizado para treinar a RNFR. Após o treinamento da RNFR, via criação ou leitura dos dados de um arquivo .trn, o robô inicia a navegação de acordo com o modelo de cenário selecionado pelo Usuário.

Caso o Usuário opte por executar o método cálculo de odometria, o robô vai realizar a navegação de acordo com o cenário escolhido pelo Usuário. Tanto na execução da navegação via método RNFR ou cálculo de odometria, enquanto o robô executa a navegação, o método selecionado tem o papel de estimar a posição do robô de acordo com a posição inicial. Cada posição estimada é enviada ao arquivo .rnfr no caso da execução do método RNFR, e ao arquivo .co caso tenha sido selecionado o cálculo de odometria. Além das posições estimadas, estes arquivos armazenam a posição real do robô no ambiente, indicada pelo Usuário.

Caso o Usuário já possua os arquivos .rnfr e .co provenientes de execuções já realizadas, ele pode selecioná-los para realizar a comparação entre os métodos de execução da navegação robótica. O protótipo deve exibir as rotas realizadas e a distância entre o ponto final estimado e o ponto real para cada um dos métodos.

Figura 8 - Diagrama de atividades



3.3 IMPLEMENTAÇÃO

A seguir são mostradas as técnicas e ferramentas utilizadas e a operacionalidade da implementação.

3.3.1 Técnicas e ferramentas utilizadas

Foram utilizadas as seguintes técnicas e ferramentas no desenvolvimento do presente trabalho:

a) implementação do software foi realizada na *Integrated Development Environment* (IDE) Eclipse através da linguagem Java;

b) framework Encog (versão 3.2.0) para desenvolvimento da RNFR;

c) JFreeChart para desenvolvimento dos gráficos que exibem as rotas executadas a partir dos métodos RNFR e cálculo de odometria;

d) LEGO Mindstorms para a criação de um robô provido de dois sensores de odometria (mais detalhes a respeito da estrutura e funcionamento no capítulo 2.3);
e) uso da biblioteca LeJOS NXT Runtime para implementação do software instalado no robô. Através desta biblioteca foi possível implementar a conexão via *bluetooth* entre o robô e o computador (mais detalhes desta biblioteca no capítulo 2.3);
f) uso da biblioteca LeJOS PC para implementação do software executado no computador que solicita a conexão via *bluetooth* ao robô (mais detalhes da biblioteca LeJOS PC no capítulo 2.3);
g) uso das funcionalidades do *Android Development Toolkit* acopladas à IDE Eclipse para desenvolvimento do software que é instalado ao aparelho celular do modelo *Sansung Galaxy* i9000. Através desse aplicativo é possível receber a conexão via *bluetooth* do computador e enviar os dados da bússola provenientes do celular;
h) biblioteca Bluecove (versão 2.1.0) que permite estabelecer uma conexão via *bluetooth* do computador para o celular. Dessa forma é possível ler os valores da bússola enviados pelo aplicativo instalado no celular.

3.3.2 Implementação da aplicação

A aplicação desenvolvida é composta de um protótipo para execução e comparação dos dados de execução da navegação robótica via RNFR e cálculo de odometria. Para realização destes métodos, o protótipo envia e recebe valores de odometria de um aplicativo instalado no robô LEGO Mindstorms desenvolvido. Além dos dados de odometria, o protótipo recebe os valores da bússola provenientes de um aplicativo executado no celular.

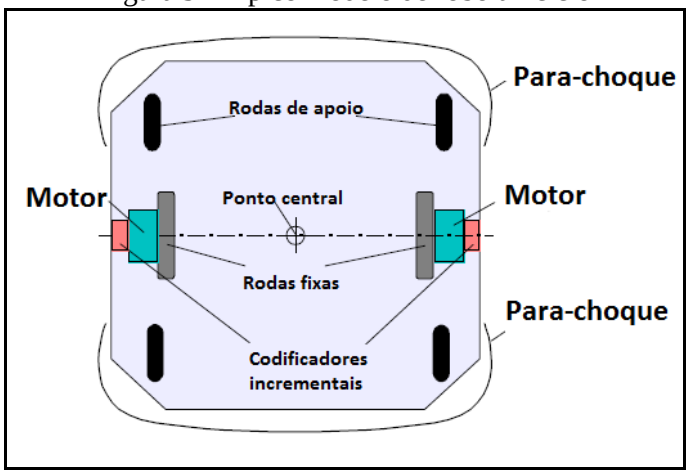
Os capítulos que se seguem têm o propósito de explicar a implementação de cada uma das partes componentes da aplicação assim como o relacionamento entre elas. Optou-se por descrever e justificar a escolha do modelo de robô construído na plataforma LEGO Mindstorms no capítulo 3.3.2.1.

3.3.2.1 Estrutura do robô LEGO Mindstorms desenvolvido

Como um dos propósitos do trabalho é o de comparar a execução da navegação através dos métodos RNFR e cálculo de odometria, foi criado um robô conforme o modelo cinemático do robô uniciclo descrito no capítulo 2.2. Isto porque, de acordo com a fundamentação apresentada, o robô uniciclo é adequado aos testes que envolvem controle de navegação robótica.

Para reproduzir o modelo cinemático do uniciclo, o robô desenvolvido é provido de duas rodas que são dispostas lateralmente ao bloco programável. Cada roda é conectada a um motor que contém o *encoder* que registra os valores de odometria do robô. Tipicamente, este robô realiza o giro baseado em seu ponto central conforme Figura 9. Para tanto seria necessário adicionar quatro rodas de apoio ao robô desenvolvido.

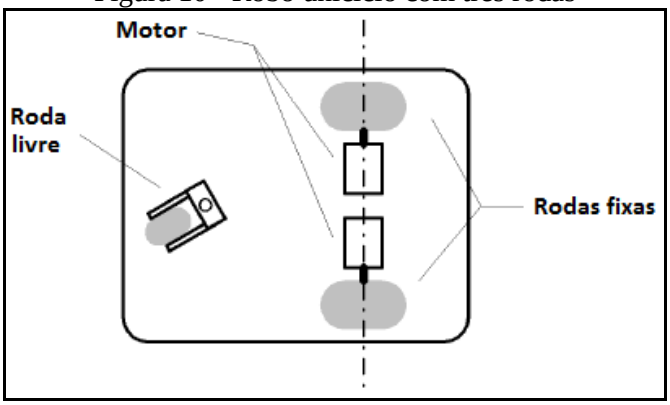
Figura 9 - Típico modelo de robô uniciclo



Fonte: baseado em Borenstein e Feng (1994, p. 3).

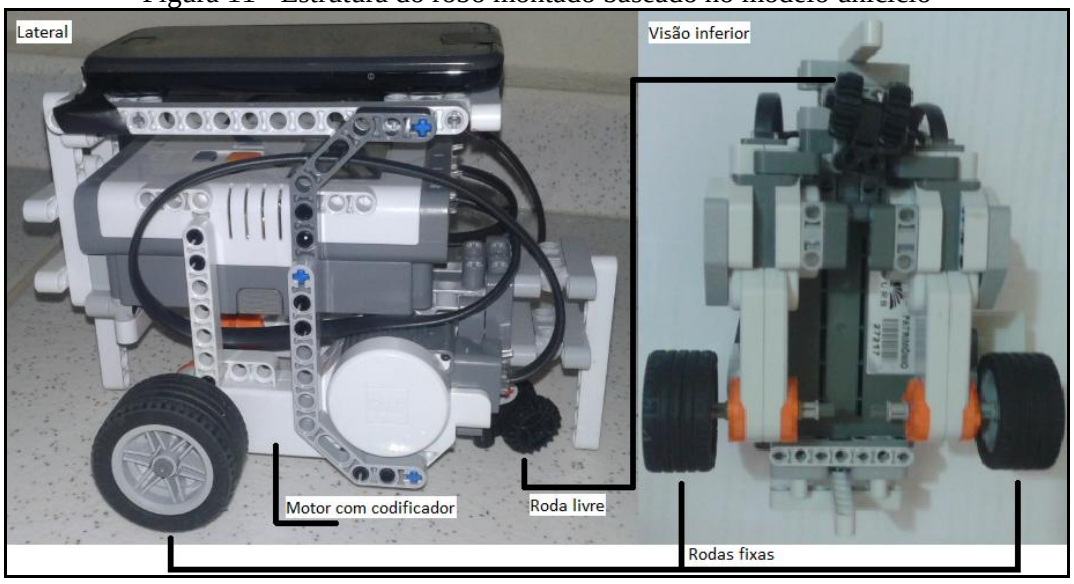
Como alternativa foi criado um robô que mantém as duas rodas fixas ligadas ao motor com funcionamento independente entre si, mas substitui as quatro rodas de apoio por uma de movimentação livre que lhe garante estabilidade, conforme proposto por Secchi (2008, p. 22) e Petrović, Ivanjko e Perić (2002, p. 92). Na Figura 10 é possível observar a semelhança entre o modelo composto por três rodas e o modelo criado no presente trabalho, conforme Figura 11.

Figura 10 - Robô uniciclo com três rodas



Fonte: baseado em Secchi (2008, p. 22).

Figura 11 - Estrutura do robô montado baseado no modelo uniciclo



Além dos dados de odometria o robô deve disponibilizar o valor de orientação através da bússola. Como o *kit* LEGO Mindstorms utilizado não dispõe desse sensor, ele foi implementado em um celular do modelo *Sansung Galaxy* i9000. Foi criada uma estrutura acima do bloco programável com o propósito de suportar o celular e não interferir o manejo dos botões de acesso ao software do robô, conforme Figura 11. Mais detalhes a respeito da montagem do robô podem ser verificados no Apêndice A.

3.3.2.2 Aplicativo executado no robô da plataforma LEGO Mindstorms

O aplicativo foi criado com o propósito de estabelecer e manter a comunicação entre o robô e o computador. Através da comunicação criada, o aplicativo tem a propriedade de enviar e ler valores do computador relacionados aos *encoders* do robô. Estes valores são utilizados para a execução dos métodos RNFR e cálculo de odometria, respectivamente.

Para realizar o desenvolvimento do aplicativo foi criado o projeto Java `SensorFusionNXT` na IDE Eclipse contendo a biblioteca LeJOS NXT Runtime. Este projeto é composto pela classe `BTReceive`, responsável por abrir e aguardar uma conexão *bluetooth*, e a classe `NXTFusion`, que contém o código que envia e lê os valores dos *encoders* a partir da conexão *bluetooth* estabelecida.

A classe `NXTFusion` é iniciada a partir do método `main` conforme o código no Quadro 6. Nas linhas 02 e 03 é realizada a chamada ao método `connect` da classe `BTReceive`. O método `connect` tem a função de abrir uma conexão *bluetooth* através da chamada do método `waitForConnection` da classe `Bluetooth` que pertence à biblioteca LeJOSNXT. O método `waitForConnection` aguarda e retorna a conexão *bluetooth* estabelecida por um dispositivo

remoto. Ao estabelecer a conexão com o aplicativo instalado ao computador através do método `connect` da classe `BTReceive`, passa a ser possível enviar e ler dados do dispositivo móvel através dos métodos `read` e `write` desta classe.

Quadro 6 - Código de inicialização da classe NXTFusion

```
01 //Chama rotina que aguarda e estabelece conexão bluetooth com o computador
02 BTReceive btReceive = new BTReceive();
03 btReceive.connect();
04
05 try {
06       while(true){
07             btReceive.write(new byte[] {(byte)rightMotor.getSpeed(),
08             (byte)leftMotor.getSpeed() });
09
10             byte[] readValues = btReceive.read();
11
12             move(readValues);
13       }
14 }finally {
15       btReceive.close();
16       System.exit(0);
17 }
```

Ainda no método `main`, na linha 06 é iniciado o ciclo de leitura e envio de dados ao dispositivo remoto conectado via *bluetooth*. As variáveis `rightMotor` e `leftMotor` são instâncias da classe `RegulateMotor` que tem acesso às propriedades do motor conectado às rodas do robô. Uma dessas propriedades é a velocidade do *encoder* que é lida através do método `getSpeed` e enviada ao dispositivo remoto através da chamada do método `write` na linha 07. Na linha 10 é realizada a leitura dos valores provenientes do dispositivo remoto, estes valores são utilizados para movimentação dos *encoders* no método `move`, conforme pode ser visualizado no Quadro 7.

Quadro 7 - Código para movimentação dos motores do robô

```
01 private static void move(byte[] readValues) throws Exception {
02       int right = byteArrayToInt(readValues, 0);
03       int left = byteArrayToInt(readValues, 4);
04       int speedRight = byteArrayToInt(readValues, 8);
05       int speedLeft = byteArrayToInt(readValues, 12);
06
07       rightMotor.setSpeed(speedRight);
08       leftMotor.setSpeed(speedLeft);
09
10       rightMotor.rotate(right, true);
11       leftMotor.rotate(left, true);
12
13       Thread.sleep(3000);
14 }
```

Os valores lidos são correspondentes à quatro das propriedades do motor: rotação do motor direito, rotação do motor esquerdo, velocidade do motor direito e velocidade do motor esquerdo. Esses valores são aplicados aos respectivos motores e, na linha 13, a execução força a espera de 3 segundos. Esse é o tempo necessário para que o robô execute a movimentação e aguarde o envio da próxima instrução oriunda do dispositivo remoto conectado.

3.3.2.3 Aplicativo executado no celular *Sansung Galaxy* i9000

O aplicativo executado no *Sansung Galaxy* i9000 tem a função de estabelecer e manter a conexão via *bluetooth* com o computador. Através desta conexão o aplicativo envia dados a respeito de sua orientação para a máquina. Como o celular fica disposto sobre o robô, o valor de sua orientação representa a orientação do robô. Este valor é utilizado, em conjunto com a velocidade dos *encoders* do robô, para criar o treinamento da RNFR.

O celular *Sansung Galaxy* i9000 não dispõe do sensor físico de bússola. No entanto ele possui outros sensores como o acelerômetro e o sensor de campo magnético. Conforme descrito em Android (2014), através desses sensores é possível emular o comportamento de uma bússola.

Para que seja possível registrar a orientação do robô durante a navegação, são enviados ao computador o valor da bússola sempre que é detectada uma alteração nos sensores de acelerômetro e campo magnético. Essas alterações são percebidas pela classe `SensorManager` que é responsável por acessar os sensores do dispositivo. Para acompanhar essas alterações sofridas pelos sensores foi necessário que a instância da classe `SensorManager` recebesse o registro de uma classe que implementa a interface `SensorEventListener`. O método `onSensorChange` pertence a esta interface e é chamado toda vez que ocorre a alteração na detecção de valores dos sensores de acelerômetro e campo magnético conforme pode ser verificado no Quadro 8.

O código existente entre as linhas 03 a 14 é responsável por realizar o registro dos valores atualizados dos sensores de acelerômetro e campo magnético. Esses valores são utilizados para calcular a matriz de rotação através do método `getRotationMatrix` na linha 15. A matriz de rotação resultante é atribuída ao parâmetro `matrixR`. A partir dos valores da matriz de rotação é possível descobrir a orientação do robô. Para tanto, a `matrixR` é passada como parâmetro do método `SensorManager.getOrientation` na linha 19. Este método retorna uma matriz que, entre outros valores, contém o azimute. O azimute representa a variação angular em torno do eixo Z (eixo que fica perpendicular à Terra). Como seu valor é calculado em radianos, na linha 21 é realizada a conversão para graus. Este valor, que representa a orientação do robô, é enviado ao dispositivo remoto conectado via *bluetooth*.

Quadro 8 - Aquisição dos valores da bússola

```
01  @Override
02  public void onSensorChanged(SensorEvent event) {
03         switch (event.sensor.getType()) {
04         case Sensor.TYPE_ACCELEROMETER:
05             for (int i = 0; i < 3; i++) {
06                valuesAccelerometer[i] = event.values[i];
07             }
08             break;
09         case Sensor.TYPE_MAGNETIC_FIELD:
10             for (int i = 0; i < 3; i++) {
11                valuesMagneticField[i] = event.values[i];
12             }
13             break;
14         }
15         boolean success = SensorManager.getRotationMatrix(matrixR,
16         matrixI,valuesAccelerometer, valuesMagneticField);
17
18         if (success) {
19             SensorManager.getOrientation(matrixR, matrixValues);
20
21             float azimuth = (float) Math.toDegrees(matrixValues[0]);
22
23             azimuth = azimuth < 0.0f ? azimuth + 360 : azimuth;
24
25             mCommandService.write(ByteBuffer.allocate(4).putInt((int)
26                azimuth).array());
27         }
28  }
```

3.3.2.4 Implementação do método de cálculo de odometria

De acordo com Petrović, Ivanjko e Perić (2002, p. 92), por conta da substituição das rodas de apoio por uma roda livre, conforme abordado no capítulo 3.3.2.1, o robô não realiza o giro baseado em seu ponto central. Se o cálculo de odometria for realizado sem considerar a alteração do comprimento efetivo do eixo, será produzido um erro sistemático. Portanto, é necessário acrescentar a variável β à fórmula que representa o valor de $\Delta\Theta_k$ (cálculo da variação de ângulo do robô em relação ao eixo x). A variável adicionada representa o parâmetro de correção do erro sistemático gerado. Por conta desta alteração o cálculo de odometria passa a ser realizado da seguinte forma (mais detalhes sobre as fórmulas e variáveis na seção 2.2):

a) $x_{k+1} = x_k + v_{totk} \cdot T \cdot \cos \Theta_{k+1}$;

b) $y_{k+1} = y_k + v_{totk} \cdot T \cdot \text{sen } \Theta_{k+1}$;

c) $\Theta_{k+1} = \Theta_k + \Delta\Theta_k$;

d) $v_{totk} = (vL_k + vR_k)/2$;

e) $\Delta\Theta_k = (180/\pi).((vR_k - vL_k)/b.\beta).T$ (fórmula que recebeu o parâmetro β).

Essas fórmulas foram implementadas no método `compute` da classe `OdometricNavigation` conforme Quadro 9.

Quadro 9 - Cálculo de odometria

```
01 public double[] compute(double centerAxleX, double centerAxleY, int
02 samplingTimeStep, double orientation, int leftSpeed, int rightSpeed) {
03
04       double variationAngleX = 0;
05
06       if (rightSpeed - leftSpeed != 0) {
07             variationAngleX = (180 / Math.PI) * ((rightSpeed -
08                   leftSpeed) / AXLE_ANGLE * CORRECTION_PARAMETER);
09       }
10
11       double nextAngle = calculateNextOrientation(orientation,
12       variationAngleX);
13
14       int totalTranslationalSpeed = (leftSpeed + rightSpeed) / 2;
15
16       double nextX = (centerAxleX + totalTranslationalSpeed *
17       samplingTimeStep * Math.cos(Math.toRadians(nextAngle)));
18
19       double nextY = (centerAxleY + totalTranslationalSpeed *
20       samplingTimeStep * Math.sin(Math.toRadians(nextAngle)));
21
22       return new double[] { nextX, nextY, nextAngle};
23 }
```

Como o cálculo de odometria exige que o ponto inicial seja conhecido, a primeira chamada desse método assume que o robô está na posição `x = 0`, `y = 0`, $\Theta = 0$. Estes valores são referentes ao eixo central do robô e no método `compute` representam as variáveis `centerAxleX`, `centerAxleY` e `orientation`, respectivamente. O fato da orientação (Θ) inicial ter valor igual à zero significa que a linha perpendicular ao ponto central entre os eixos esquerdo e direito necessariamente deve estar sobre o eixo `x`.

Na linha 06 do Quadro 9, o primeiro valor calculado é a variação de ângulo do robô em relação ao eixo `x`. Entende-se que caso o robô tenha as mesmas velocidades no motor esquerdo e direito, não deve existir variação angular já que, inicialmente, ele estava sobre o eixo `x`. Caso contrário, é necessário calcular a variação angular, conforme realizado na linha 07. Este valor é utilizado para calcular o posicionamento do robô no momento subsequente, portanto para estimar os valores de x_{k+1}, y_{k+1} e Θ_{k+1}.

O valor de Θ_{k+1} é calculado na linha 13 através da chamada do método `calculateNextOrientation` que recebe como parâmetros a orientação atual e a variação angular. A implementação deste método pode ser verificada no Quadro 10.

Quadro 10 - Cálculo da orientação subsequente

```
01  private double calculateNextOrientation(double actualOrientation,
02          double variation) {
03          double nextOrientation = actualOrientation + variation;
04
05          if (nextOrientation < 0) {
06                  return 360 + nextOrientation;
07          }
08          if (nextOrientation > 360) {
09                  return nextOrientation - 360;
10          }
11          return nextOrientation;
12  }
```

O próximo ângulo é a soma do ângulo atual mais a variação angular. No entanto, são permitidos valores de 0° à 360°. Por esse motivo, se por exemplo o cálculo realizado na linha 04 do Quadro 10 resultar em -6, a próxima orientação será de 354° conforme tratamento da linha 07, mas se resultar em 365, a variável `nextOrientation` vai receber valor de 5° de acordo com a linha 10.

Após ser calculado o valor de Θ_{k+1} é verificada a velocidade total de translação do robô autônomo representada pela variável `totalTranslationalSpeed` na linha 14 do Quadro 9. Este cálculo depende da velocidade dos *encoders* acoplados aos motores direito e esquerdo do robô autônomo. Portanto, antes de chamar o método `compute`, são obtidos os valores de velocidade a partir da conexão estabelecida com o aplicativo executado no robô da plataforma LEGO Mindstorms (mais detalhes no capítulo 3.3.2.2).

Por fim, nas linhas 16 e 19 do Quadro 9, estão os cálculos de x_{k+1} e y_{k+1}, respectivamente. Estes cálculos fazem uso das variáveis já calculadas e do parâmetro `samplingTimeStep` que representa o intervalo entre as chamadas do método `compute`. Neste trabalho, o valor da variável `samplingTimeStep` sempre recebe 1 já que o método `compute` é continuamente chamado. Alterando ou não a velocidade, o método `compute` deve ser chamado novamente para estimar a nova posição. A cada nova execução, os valores de x_{k+1}, y_{k+1} e Θ_{k+1} calculados na chamada anterior, devem ser passados como os parâmetros `centerAxleX`, `centerAxleY` e `orientation`, respectivamente. Este ciclo é realizado até que a posição de destino desejada seja alcançada. Além dos valores da posição serem utilizados para a correção da guiagem robótica, eles são gravados em um arquivo com extensão `.co` que é utilizado para realizar a comparação entre os métodos de cálculo de odometria de RNFR.

3.3.2.5 Implementação do método RNFR

Conforme mencionado no capítulo 2.2 o cálculo de odometria não leva em consideração os erros sistemáticos. Isto pode ser verificado no capítulo anterior (3.3.2.4) em que a variação angular é considerada igual a zero se a velocidade entre as rodas for igual. No

entanto, caso o robô navegue com a mesma velocidade dos *encoders* esquerdo e direito mas o diâmetro das rodas seja diferente, pode haver uma variação angular. Este erro será acumulado durante o cálculo de odometria e acarretará na imperícia da guiagem robótica que depende da posição estimada.

A RNFR implementada neste trabalho tem a função de associar o valor da velocidade dos *encoders* esquerdo e direito do robô autônomo com o valor de orientação. Esta implementação utiliza como base o trabalho desenvolvido por Petrović, Ivanjko e Perić (2002, p. 93) e propõe que a variação angular seja calculada através da estrutura exibida na Figura 12. Portanto, o resultado obtido através dessa RNA pode substituir a fórmula que calcula o valor de $\Delta\Theta_k$ no cálculo de odometria (detalhes na seção 3.3.2.4). Logo, o cálculo de estimação da posição robótica (x_{k+1}, y_{k+1} e Θ_{k+1}) passa a ser realizado da seguinte forma:

a) $x_{k+1} = x_k + v_{totk} \cdot T \cdot \cos \Theta_{k+1}$;

b) $y_{k+1} = y_k + v_{totk} \cdot T \cdot \text{sen } \Theta_{k+1}$;

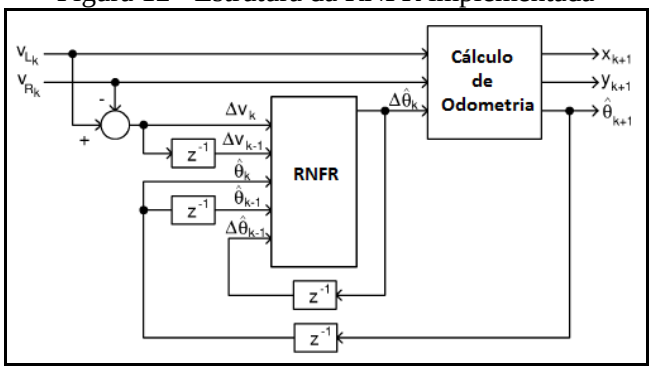
c) $\Theta_{k+1} = \Theta_k + \Delta\Theta_k$;

d) $v_{totk} = (vL_k + vR_k)/2$;

e) $\Delta\Theta_k$ = (valor obtido a partir da RNA implementada).

A RNFR proposta possui cinco entradas: a diferença da velocidade entre o eixo esquerdo e direito, a diferença da velocidade entre o eixo esquerdo e direito na iteração anterior, a orientação atual, a orientação no passo anterior e a variação angular da iteração anterior. Conforme mencionado, a saída representa a variação angular e é utilizada, conforme pode ser verificado na Figura 12, para realizar o cálculo de odometria.

Figura 12 - Estrutura da RNFR implementada

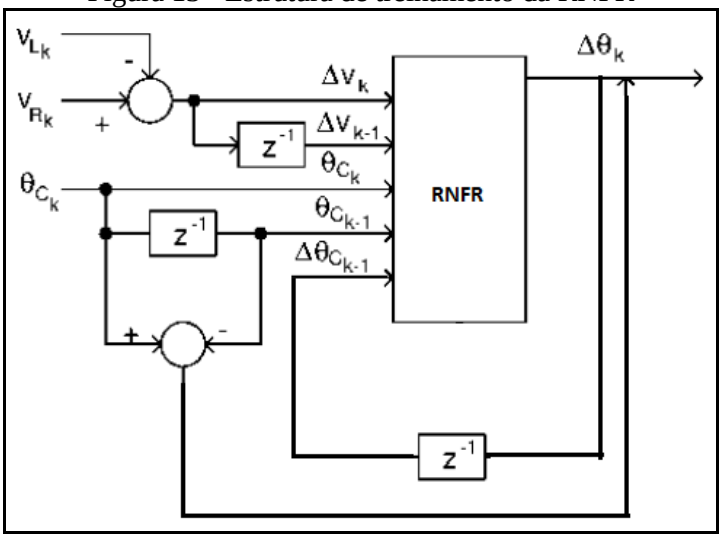


Fonte: baseado em Petrović, Ivanjko e Perić (2002, p. 93).

Para que a RNA estime o valor correto de variação angular é necessário que ela seja treinada para capturar os erros associados à orientação do robô autônomo. Este treino é realizado antes da execução da RNFR. Recebe um vetor de entrada composto dos valores de diferença entre a velocidade dos *encoders* direito e esquerdo do robô, diferença entre a

velocidade dos *encoders* direito e esquerdo do robô na iteração anterior, orientação, orientação na iteração anterior e a variação de orientação na iteração anterior medidos através da bússola acoplada ao robô. Como saída, a estrutura recebe um vetor contendo os valores da variação angular correspondentes às entradas apresentadas. O uso dos valores combinados de bússola e da velocidade dos *encoders* representa o processo de FS realizado para melhorar a acurácia da leitura da variação angular do robô autônomo. Esta estrutura pode ser visualizada na Figura 13.

Figura 13 - Estrutura de treinamento da RNFR



Fonte: baseado em Petrović, Ivanjko e Perić (2002, p. 94).

No código fonte, os valores que compõe os vetores de entrada e saída do treinamento são adquiridos a partir da chamada do método `getSample` da classe `RadialBasisFunctionNN`. Este método recebe como valores de entrada a velocidade dos *encoders* esquerdo e direito por intermédio do aplicativo executado no robô da plataforma LEGO Mindstorms (mais detalhes na seção 3.3.2.2) e a orientação obtida a partir do aplicativo executado no celular Sansung Galaxy i9000 (mais detalhes na seção 3.3.2.2). A implementação desse método pode ser verificada no Quadro 11.

Quadro 11 - Aquisição de dados para treinamento da RNFR

```
01  private Sample getSample(int leftSpeed, int rightSpeed, double angle) {
02
03          double speedDif = rightSpeed - leftSpeed;
04
05          double angleChange = calculateAngleChange(pvAngle, angle);
06
07          // valores padronizados
08          double stdSpeedDif = standardize(speedDif);
09          double stdPvSpeedDif = standardize(pvSpeedDif);
10          double stdAngle = standardize(angle);
11          double stdPvAngle = standardize(pvAngle);
12          double stdPvAngleChange = standardize(pvAngleChange);
13          double stdAngleChange = standardize(angleChange);
14
15          Sample sample = createSample(stdSpeedDif, stdPvSpeedDif, stdAngle,
16          stdPvAngle, stdPvAngleChange, stdAngleChange);
17
18          pvAngleChange = angleChange;
19          pvAngle = angle;
20          pvSpeedDif = speedDif;
21
22          return sample;
23  }
```

Na linha 13 o valor de variação angular é calculado a partir do método `calculateAngleChange` que retorna a diferença entre o ângulo atual e o ângulo lido na iteração anterior. Nas linhas 08 à 13, os valores adquiridos são padronizados através do método `standardize` que recebe o valor lido dos *encoders* ou bússola e os transforma na escala que varia de +1 à -1. Esta operação deve ser realizada para auxiliar na diminuição do erro do treinamento da RNFR.

De acordo com Petrović, Ivanjko e Perić (2002, p. 94), para que esta RNA tivesse uma boa assertividade o treinamento deveria ser realizado de forma a cobrir o maior número de combinações possíveis de velocidade e orientação considerando os cenários a serem percorridos pelo robô. No entanto, tantas combinações geraram uma infinidade de erros de estimação na RNA. Por esse motivo a rede foi treinada apenas executando uma reta e um giro de 90°. Dessa forma, ao movimentar o robô são acrescidas à navegação apenas as alterações de orientação recorrentes ao sentido que o robô se encontra. Os valores adquiridos a partir das chamadas do método `getSample` são salvos em um arquivo com extensão `.trn` que é utilizado posteriormente para treinar a RNFR sem que seja necessário gerar novamente a sequência de treinamento. Esta sequência de treinamento deve ser chamada novamente caso seja alterada a estrutura do robô, já que podem existir erros sistemáticos não identificados no treinamento gerado a partir de outro modelo.

Os valores contidos no arquivo com extensão `.trn` são utilizados para treinar a RNA. No Quadro 12 é possível verificar a estrutura da RNFR desenvolvida no *framework* Encog e o código relacionado à execução do seu treinamento.

Quadro 12 - Criação e treinamento da RNFR

```
01  @Override
02  public void calibrate() {
03
04        int dimensions = 5;
05        int numNeuronsPerDimension = 2;
06        double volumeNeuronWidth = 2.0 / numNeuronsPerDimension;
07
08        int numNeurons = (int) Math.pow(numNeuronsPerDimension, dimensions);
09
10        RadialBasisPattern pattern = new RadialBasisPattern();
11        pattern.setInputNeurons(dimensions);
12        pattern.addHiddenLayer(numNeurons);
13        pattern.setOutputNeurons(1);
14        network = (RBFNetwork) pattern.generate();
15
16        network.setRBFCentersAndWidthsEqualSpacing(-1, 1, RBFEnum.Gaussian,
17        volumeNeuronWidth, true);
18
19        if (!loadTraining) {
20              training(samplesNumber);
21        } else {
22              training(trainingFile, samplesNumber);
23        }
24
25        BasicMLDataSet trainingSet = new BasicMLDataSet(inputTraining,
26        expectedTraining);
27
28        MLTrain train = new SVDTraining(network, trainingSet);
29        int epoch = 1;
30        do {
31              train.iteration();
32              epoch++;
33        } while ((epoch < 1) && (train.getError() > 0.001));
34  }
```

Na linha 10 é criado um objeto do tipo `RadialBasisPattern`, classe que pertence ao *framework* Encog, e tem por função montar a estrutura da RNFR. Esta estrutura é composta por uma camada de entrada, uma camada oculta e uma da saída declaradas nas linhas 11, 12 e 13, respectivamente. A camada de entrada tem o tamanho de 5 neurônios correspondentes aos valores de entrada indicados na Figura 12. O tamanho da camada escondida foi baseado no trabalho de Petrović, Ivanjko e Perić (2002, p. 95), em que são utilizados 45 neurônios. Uma das limitações do *framework* Encog é de aceitar apenas potências do número de neurônios de entrada. Por esse motivo, a camada escondida não tem 45 neurônios mas sim 32.

A RNFR gerada na linha 14 tem os pesos, centros e a função Gaussiana associados através da função `setRBFCentersAndWidthsEqualSpacing` na linha 16. Na linha 19 é verificado se o `usuário` pretende gerar um arquivo `.trn` ou apenas carregá-lo. Independente da escolha, o método training carregará os valores de treinamento para os vetores de entrada `inputTraining` e `expectedTraining`, respectivamente. Esses vetores são utilizados para realizar o treinamento da RNFR que, diferentemente das demais RNAs, ocorre em apenas uma iteração. O código da execução do treinamento pode ser verificado entre as linhas 30 e 33 do Quadro 12.

Após a RNFR estar treinada é possível utilizá-la para gerar o valor de variação angular. Para tanto, a classe `RadialBasisFunctionNN`, assim como a classe `OdometricNavigation`, também possui o método `compute` apresentado no Quadro 9. A diferença entre os métodos dessas duas classes é em relação ao cálculo da variação angular. A variável `variationAngleX` recebe o valor calculado pela RNFR na linha 10 do Quadro 13. Para que a RNFR calcule a variação angular, ela recebe os valores do estado atual do robô autônomo calculados através do método `getCurrentState`. Este método gera as entradas da RNFR conforme o modelo exibido na Figura 12. O valor da variação angular calculado na linha 10 é submetido ao método `reverseStd` que tem a função de reverter a padronização realizada pelo método `standardize` no momento da aquisição dos dados para treinamento (mais detalhes no Quadro 11).

Quadro 13 - Posição estimada através de RNFR

```
01  public double[] compute(double centerAxleX, double centerAxleY, int
02  samplingTimeStep, double orientation, int leftSpeed, int rightSpeed) {
03
04        Sample currentState = getCurrentState(leftSpeed, rightSpeed,
05        orientation);
06
07        computedData.clear();
08        computedData.setData(currentState.getInputValues());
09
10        MLData compute = network.compute(computedData);
11        double variationAngleX = reverseStd(compute.getData()[0]);
12        double nextAngle = calculateNextOrientation(orientation,
13        variationAngleX);
14
15        int totalTranslationalSpeed = (leftSpeed + rightSpeed) / 2;
16
17        double nextX = (centerAxleX + totalTranslationalSpeed *
18        samplingTimeStep * Math.cos(Math.toRadians(nextAngle)));
19
20        double nextY = (centerAxleY + totalTranslationalSpeed *
21        samplingTimeStep * Math.sin(Math.toRadians(nextAngle)));
22
23        return new double[] { nextX, nextY, nextAngle};
24  }
```

3.3.3 Operacionalidade da implementação

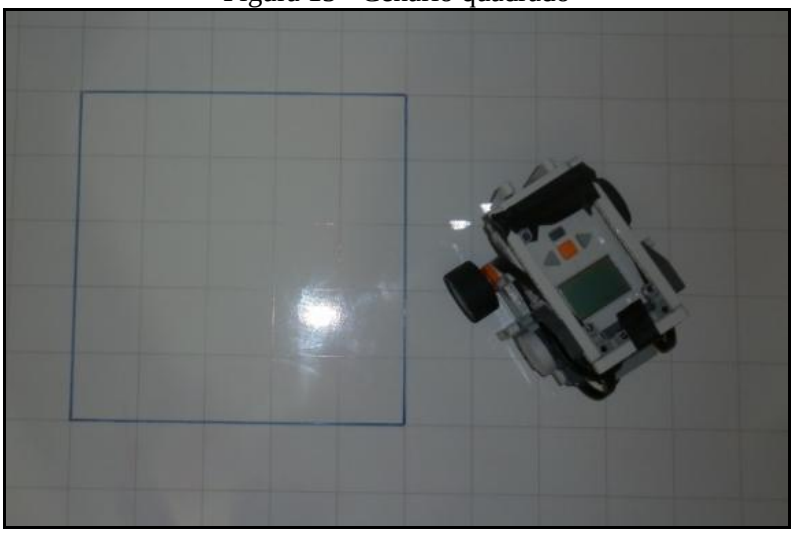
O protótipo para comparação dos métodos de execução de navegação é inicializado através do método `main` da classe `Prototype`. A partir desde protótipo são realizadas as aquisições e comparação dos dados extraídos a partir dos métodos de RNFR e cálculo de odometria conforme Figura 14.

Figura 14 - Tela de execução e comparação dos métodos de navegação

3.3.4 Execução dos métodos de navegação robótica

Para realizar a execução de um método de navegação robótica é necessário selecionar o cenário que o robô vai navegar. De acordo com a Figura 14 o cenário selecionado é o `Quadrado Horário`. Este cenário representa fisicamente um quadrado de 25 cm^2 desenhado sobre um quadro branco quadriculado conforme a Figura 15. A execução deste cenário consiste em o robô partir do ponto inicial, executar três rotações de 90° e retornar ao ponto inicial.

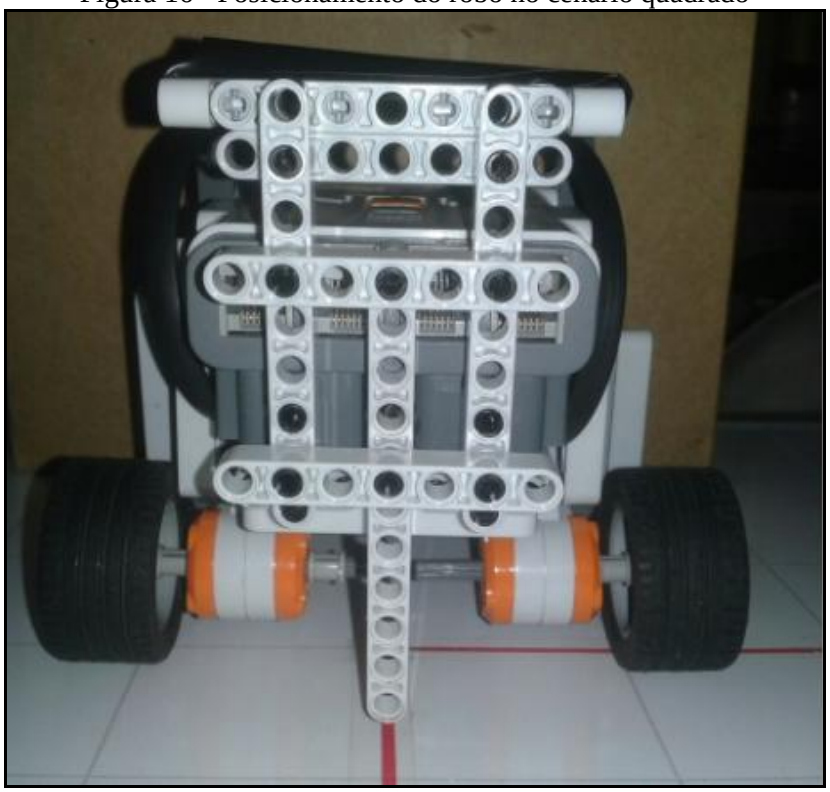
Figura 15 - Cenário quadrado



Antes de clicar no botão `Executar`, é necessário posicionar o robô no cenário. Conforme Figura 16 o robô possui um pino na vertical que é utilizado como ponto de referência para posicioná-lo sobre o eixo `x`. Os centros dos eixos devem ser colocados sobre o eixo `y`. Após posicionar o robô, o usuário deve liga-lo e executar o arquivo `NXTFusion` que

inicializa o aplicativo descrito na seção 3.3.2.2. Este aplicativo aguarda a conexão via *bluetooth* proveniente do computador para que comece a enviar e ler os valores dos *encoders*. Nesse momento, o usuário deve clicar no botão `Executar` que inicia a conexão *bluetooth* requisitada.

Figura 16 - Posicionamento do robô no cenário quadrado



Considerando que o usuário selecionou o valor `Cálculo de Odometria` no campo `Método de Execução` do protótipo, ao clicar no botão `Executar` é realizada a chamada do método compute conforme código demonstrado no Quadro 9. Como a estimação da posição odométrica deve partir de uma posição conhecida, esse método é chamado com as variáveis `centerAxleX`, `centerAxleY` e `orientation`, valendo 0. Por esse motivo, é importante posicionar corretamente o robô no cenário físico de forma que a perpendicular entre os eixos fique sobre o eixo `x` ($\Theta=0$), e o pino vertical esteja sobre o ponto inicial ($y=0$ e $x=0$).

A chamada do método `compute` retorna a posição estimada (Θ_{k+1}, y_{k+1} e x_{k+1}). Este valor tem três destinos: primeiramente ele é renderizado no protótipo, depois ele é adicionado ao arquivo de extensão `.co`, por fim, ele é utilizado para verificar se o robô está em uma posição de acordo com o cenário selecionado. Se o robô não estiver na posição correspondente ao esperado pelo cenário, são alterados os valores dos *encoders* esquerdo e direito de forma que ele passe a ter a posição ajustada. Neste trabalho, os ajustes de guiagem se limitam a verificar se o robô chegou no limite do cenário e executar o movimento de rotação até alcançar o

ângulo desejado. No caso do cenário `Quadrado Horário` a rotina verifica se a posição estimada pelo método `compute` alcançou o limite do quadrado. Caso essa situação seja verificada a rotina executa o giro do robô até que a posição estimada esteja acrescida de 90°.

No fim da execução do cenário, é apresentada uma tela ao usuário solicitando que ele insira o valor de `x` e `y` referentes à posição final no cenário físico. Estes valores também são adicionados ao arquivo `.co` para posterior realização da comparação. O resultado da execução pode ser visualizado no protótipo conforme Figura 17.

Figura 17 - Execução do método de cálculo de odometria

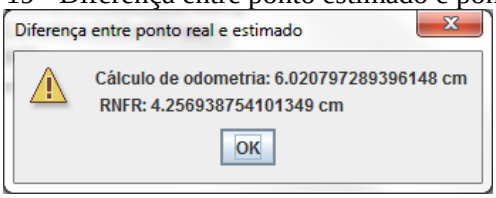
Para realizar a execução do método `RNFR` devem ser seguidos os mesmos passos descritos para a execução do `Cálculo de Odometria`. A diferença é que a RNA deve ser treinada antes que o método seja executado. Para tanto, é solicitado ao usuário que selecione um arquivo `.trn` ou aguarde o robô realizar a rotina de treinamento. Além disso, as posições são estimadas pelo método `compute` da classe `RadialBasisFunctionNN` ao contrário do método anterior que executa o método `compute` da classe `OdometricNavigation`.

3.3.5 Comparação das execuções

Para comparar as execuções é necessário indicar um arquivo de extensão `.co` no campo `Cálculo de Odometria` do protótipo, e um arquivo `.rnfr` no campo `RNFR`, e clicar no botão `Comparar`. O resultado da comparação é exibido conforme Figura 18. Além da exibição gráfica da diferença de execução entre os métodos, é mostrada ao usuário a distância entre o ponto final estimado pelo método de cálculo de odometria e o ponto real. O mesmo é feito para o método RNFR, conforme Figura 19.

Figura 18 - Comparação entre o cálculo de odometria e RNFR

Figura 19 - Diferença entre ponto estimado e ponto real



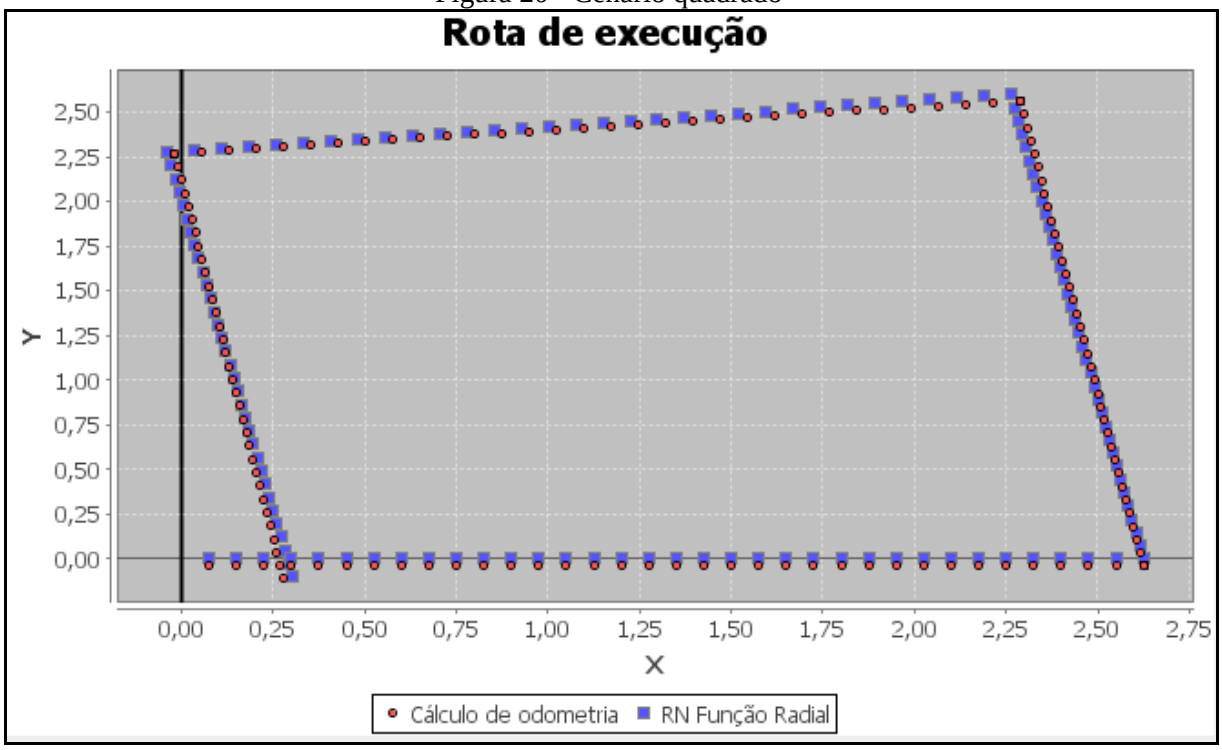
3.4 RESULTADOS E DISCUSSÃO

O cenário de teste desenvolvido para essa aplicação é baseado no primeiro trabalho correlato e nos autores Borenstein e Feng (1994, p.4). O propósito desse cenário é verificar a capacidade de estimativa de posição robótica em diferentes situações.

O cenário executado é um quadrado de 26 cm^2. Na Figura 20 é possível verificar a comparação dos métodos de execução robótica RNFR e cálculo de odometria para o cenário mencionado. Conforme requisito dos métodos desenvolvidos, o robô foi posicionado inicialmente em `x=0`, `y=0` e `Θ=0`. Neste cenário implementado, a movimentação do robô é realizada da seguinte forma: o robô deve percorrer em linha reta a extensão de 26 cm, após este ponto ele deve rotacionar enquanto não tiver acrescido 90° a partir do ponto em que se encontra. Essa ação deve ser realizada três vezes de forma a completar o percurso. O quadrado de 26 cm^2 representa apenas uma referência física da movimentação do robô. O objetivo do cenário não é que o robô realize a rota sobre o quadrado de forma perfeita, mas sim que os métodos de RNFR e cálculo de odometria sejam capazes de estimar a posição percorrida.

Na Figura 20 é possível verificar que há pouca diferença entre a execução do cálculo de odometria e do método RNFR. No caso do robô desenvolvido, o método de RNFR não reconhece variação de angulação para quando os *encoders* possuem a mesma velocidade. Essa variação também não é reconhecida pelo cálculo de odometria. No entanto, caso essa variação existisse, o método do cálculo de odometria não verificaria a alteração já que considera apenas a velocidade dos *encoders* e não uma possível variação angular.

Figura 20 - Cenário quadrado



O primeiro aspecto a ser considerado na execução desses cenários é que, ao contrário do segundo trabalho correlato, eles são inferiores a 1 metro enquanto que os correlatos trabalham com cenários de 5 metros. Por conta desse motivo, os erros encontrados no trabalho correlato são proporcionalmente maiores que os relatados nesse trabalho. No entanto, mesmo em cenários possuindo tamanho reduzido é possível constatar o resultado do acúmulo de erros sistemáticos do robô montado.

Foi verificado que o método RNFR possui um desempenho bastante similar ao método do cálculo de odometria. No entanto, caso o treinamento seja bem realizado, assim como o posicionamento inicial do robô seja preciso, é possível detectar pequenas alterações de orientação através do método de RNFR. No entanto, se o treinamento não possui as entradas requisitadas, ou se o posicionamento inicial foi mal efetuado, ocorre do cálculo de odometria ter desempenho superior ao método desenvolvido.

A diferença entre o ponto estimado e o real é pequena para ambos os métodos, mas é importante lembrar que esses cenários medem erros sistemáticos, portanto quanto maior o percurso, maior a quantidade de erros acumulados. Nesses casos, para que a navegação seja executada adequadamente, é essencial ter o conhecimento preciso da posição estimada para que seja possível realizar as correções de guiagem necessárias. É importante salientar que este trabalho não tem o propósito de desenvolver técnicas de guiagem robótica, mas sim de disponibilizar um método relativamente simples e eficaz para estimar a posição robótica a partir de um ponto conhecido.

O uso da RNFR, que teve sua estrutura fortemente baseada no primeiro trabalho correlato assim como o entendimento adquirido a partir do terceiro trabalho correlato, possui como vantagem não necessitar o uso constante de sensores adicionais. A bússola foi utilizada apenas durante o treinamento para realizar o processo de FS entre os valores de orientação e velocidade dos *encoders*. Através da realização deste treinamento foi possível captar os erros sistemáticos inerentes ao robô desenvolvido. Após a realização do treinamento o robô pode executar o método RNFR sem o uso da bússola pois a RNA já possui a capacidade de compreender a relação entre a velocidade dos *encoders* e a orientação resultante. A grande vantagem deste método de treinamento reside no fato de que, se o robô tiver a estrutura modificada, basta executar a rotina de treinamento para que a RNA reconheça os novos erros sistemáticos.

Apesar da vantagem em utilizar menos sensores no método de RNFR, os resultados demonstraram uma diferença mínima entre os métodos. Além da diferença ser pouco notável, possui a dependência de um bom treinamento e posicionamento inicial no cenário. Portanto, embora o método RNFR represente um método capaz de estimar a posição robótica, ele poderia ser melhorado caso os valores da bússola fossem lidos diretamente do sensor e não dependessem puramente do treinamento.

A principal limitação deste trabalho está no fato de o método RNFR não detectar erros não sistemáticos, como na ocorrência de derrapagens por consequência de imperfeições no chão, conforme é realizado no segundo trabalho correlato. A solução apresentada no segundo trabalho correlato para correção de erros não sistemáticos, no caso através do uso de uma câmera com visão global, não seria a solução mais adequada ao presente trabalho. Isto porque o software que reconhece as imagens obtidas pela câmera necessita conhecer o espaço para que possa estimar a posição e realizar as correções de guiagem necessárias. Esta ação de reconhecer o ambiente para então estimar a posição é caracterizada como uma estimação de posição absoluta. Embora seja mais adaptável ao mundo real este software possui custos mais

elevados de manutenção se comparado com um método que realiza a estimação da posição a partir de um ponto conhecido (mais detalhes no capítulo 2.1.2). No entanto, existem outros métodos, conforme descrito em Borenstein e Feng (1994, p. 27), para reconhecimento e tratamento de erros não sistemáticos para quando é executada a estimação da posição relativa, caso dos métodos do cálculo de odometria e RNFR.

Para melhor compreender a relação entre o trabalho atual e os trabalhos correlatos, a seguir é apresentado o Quadro 14 que contém um comparativo entre as características existentes.

Quadro 14 - Características dos trabalhos correlatos

Característica	**Petrović, Ivanjko e Perić (2002)**	**MARÍN et al. (2013)**	**Faceli (2001)**	**Trabalho atual**
Implementação do método de FS RNFR			X	X
Uso da RNFR para estimação de posição	X			X
Implementação do método de FS Filtro de Kalman	X			
Estimação de posicionamento robótico	X	X		X
Uso do *kit* LEGO Mindstorms		X		X
Implementação do cálculo de odometria	X			X
Comparação entre cálculos de estimação de posição robótica	X			X
Cálculo de posicionamento relativo	X			X

4 CONCLUSÕES

Este trabalho apresentou a implementação e comparação dos métodos de execução de navegação robótica RNFR e cálculo de odometria. A escolha desses métodos não foi realizada ao acaso. Antes de decidir por sua implementação foi necessário estudar a fundamentação teórica relacionada aos problemas encontrados na navegação robótica.

Primeiramente, foi essencial compreender os diferentes métodos de estimação de posição para que fosse possível estabelecer um comparativo entre essas soluções. Esta etapa foi necessária porque para desenvolver uma solução de estimação do posicionamento robótico é necessário verificar as ferramentas disponíveis. Primeiramente, foi considerada a natureza das informações obtidas pelos sensores disponíveis como também suas características de precisão, repetibilidade na aquisição de valores e complexidade na implementação do software para aquisição das medições. Um segundo aspecto de importância é o custo de manutenção da solução desenvolvida, pois muitas soluções apresentam resultados bastante satisfatórios na estimação da posição, mas dependem do uso de sensores com custo elevado. Além desses aspectos, foi analisada a facilidade de uso da solução implementada pois, especialmente em trabalhos que estimam a posição absoluta, é necessário realizar o mapeamento do ambiente, ou utilizar rotinas de tratamento de imagem que são dependentes de luminosidade, ou são cenários extremamente extensos por conta da precisão dos sensores utilizados.

A análise de todas essas variáveis foi auxiliada pelos trabalhos correlatos selecionados e também através das referências citadas durante o trabalho. Foi verificado que a navegação baseada na estimação da posição relativa possui o custo, manutenção e complexidade de utilização reduzidos. Que esta solução em geral é baseada no cálculo de odometria (técnica de DR) e possui um custo reduzido, mas que apresenta acúmulo ilimitado de erros de estimação. Frente à essa situação, este trabalho apresentou a implementação do método de FS RNFR que tem o propósito auxiliar o cálculo de odometria pois considera variações de orientação.

A partir do cenário de teste é possível verificar a aplicabilidade do método de FS RNFR para a estimação da posição robótica já que a posição estimada é bastante próxima a posição real. Por esse motivo considera-se que o objetivo do trabalho foi alcançado. No entanto, é importante mencionar que a sua aplicabilidade em cenários que exijam precisão de navegabilidade poderia ser melhorada se o método de RNFR também tratasse os erros não sistemáticos.

4.1 EXTENSÕES

São sugeridas as seguintes extensões:

a) implementação do tratamento de erros não sistemáticos;
b) comparação do método de RNFR com outros métodos de aprimoramento de estimação da precisão robótica;
c) aplicar este trabalho na execução de cenários mais complexos;
d) adicionar valor da bússola de forma recorrente à execução da RNFR;
e) repassar a execução realizada no computador para o celular (diminuição da integração de hardware).

REFERÊNCIAS

ANDROID. **Sensors overview**. [S.l.], 2014. Disponível em: <http://developer.android.com/guide/topics/sensors/sensors_overview.html>. Acesso em: 10 set. 2014.

BORENSTEIN, Johann; EVERETT, Bart H. R.; FENG, Liqiang. **Where am I? sensors and methods for mobile robot positioning**. Ann Arbor: Ed. da Universidade de Michigan, 1996.

BORENSTEIN, Johann; FENG, Liqiang. **UMBmark - a method for measuring, comparing, and correcting dead-reckoning errors in mobile robots**. 1994. Relatório de pesquisa n. UM-MEAM-94-22 arquivado na Universidade de Michigan, Ann Arbor.

BOWDITCH, Nathaniel. **The american practical navigator:** An Epitome of Navigation. 9nd ed. [S.l.]: Nabu Press2002.

COUTO, Leandro N. **Sistema para localização robótica de veículos autônomos baseado em visão computacional por pontos de referência**. 2012. 82 f. Dissertação (Mestrado em Ciências da Computação e Matemática Computacional) - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos.

FACELI, Katti. **Combinação de métodos de inteligência artificial para fusão de sensores**. 2001. 174 f. Dissertação (Mestrado em Ciências da Computação e Matemática Computacional) - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos.

LEGO GROUP. **Mindstorms**. [S.l.], 2014. Disponível em: <http://mindstorms.lego.com/en-us/Default.aspx>. Acesso em: 12 mar. 2014.

LEJOS. **LeJOS, Java for lego mindstorms**. [S.l.], 2014. Disponível em: <http://lejos.sourceforge.net/>. Acesso em: 12 mar. 2014.

LIGGINS, Martin. E.; HALL, David. L.; LLINAS, James. **Handbook of multisensor data fusion:** theory and practice. 2nd ed. London: CRC Press, 2009.

MARÍN, Leonardo et al. Multi sensor fusion framework for indoor-outdoor localization of limited resource mobile robots. **Sensors**, Valência, v. 26, n. 13, p. 14133-14160, Aug. 2013.

NASCIMENTO, Rafaela C. A. **Localização de robôs móveis em ambientes fechados em tempo real utilizando câmeras montadas no teto**. 2014. 53 f. Dissertação (Mestrado em Engenharia de Computação) - Programa de Pós-Graduação em Engenharia Elétrica e de Computação, Universidade Federal do Rio Grande do Norte, Natal.

PEDROSA, Diogo P. F. **Sistema de navegação para robôs móveis autônomos**. 2001. 85 f. Dissertação (Mestrado em Engenharia Elétrica) - Centro de Tecnologia, Universidade Federal do Rio Grande do Norte, Natal.

PETROVIĆ, Ivan; IVANJKO, Edouard; PERIĆ, Nedjeljeko. Neural network based correction of odometry errors in mobile robots. In: FIRA CONGRESS, 2002, Seoul. **Proceddings...** Seoul: FIRA, 2002. p. 91-96.

RUSSEL, Stuart J.; NORVIG, Peter. **Artificial intelligence:** a modern approach. 2nd ed. New Jersey: Prentice Hall, 2004.

SALUSTIANO, Rogério E. **Aplicação de técnicas de fusão de sensores no monitoramento de ambientes**. 2006. 162 f. Dissertação (Mestrado em Engenharia Elétrica) - Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas, Campinas.

SANDI, Franz A.; HEMERLY, Elder M.; LAGES, Walter F. Sistema para navegação e guiagem de robôs móveis. **SBA Controle & Automação**, [S.l], v. 9, n. 3, p. 107-118.

SECCHI, Humberto A. **Uma introdução aos robôs móveis**. 2008. 91f. Dissertação (Mestrado em Engenharia Autônoma) - Instituto de Automática, Universidade Nacional de San Juan, San Juan.

SHAREEF, Ali et al. Comparison of mlp neural network and kalman filter for localization in wireless sensor networks. In: INTERNATIONAL CONFERENCE PARALLEL AND DISTRIBUTED COMPUTING AND SYSTEMS, 2007, Cambridge. **Proceddings...** Cambridge: IASTED, 2007. p. 323-330.

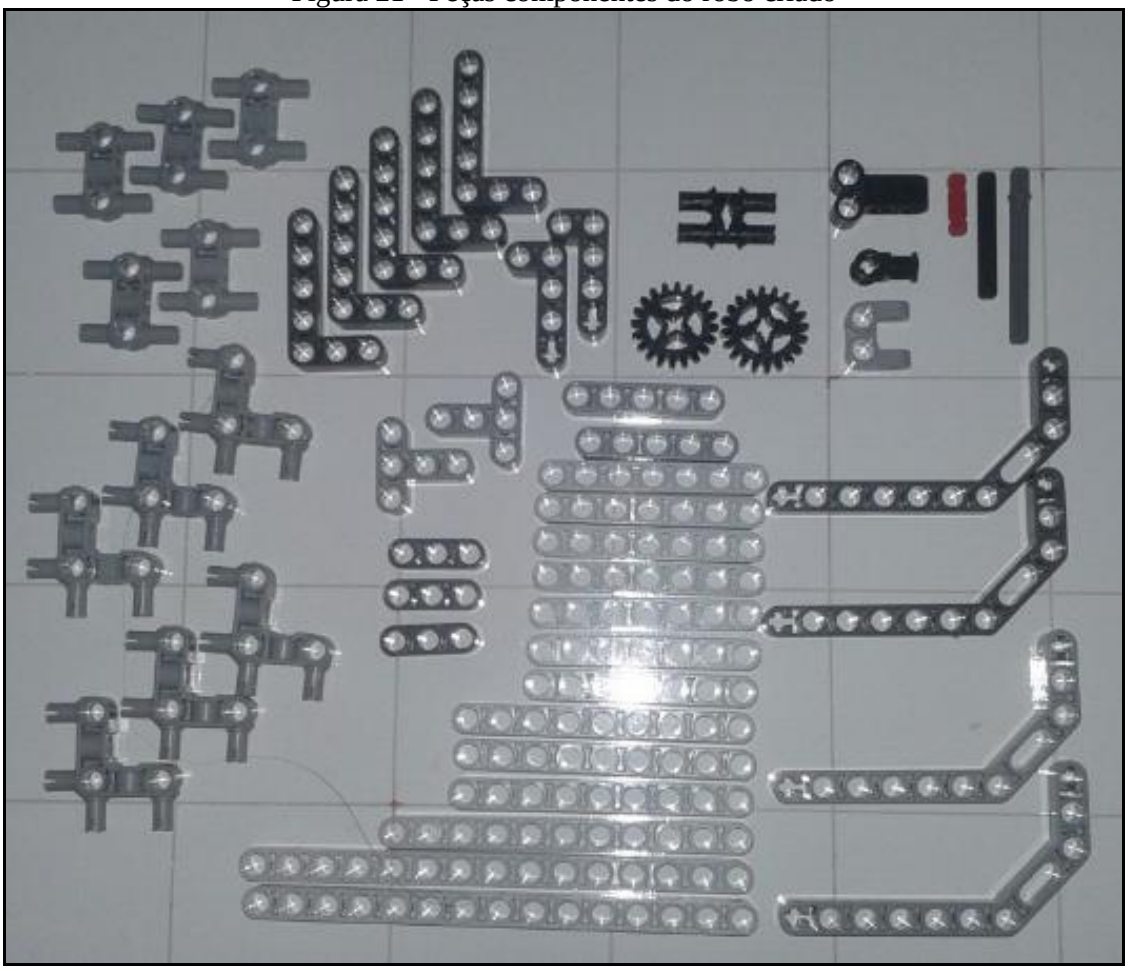
SUEGWART, Roland; NOURBAKHSH, Illah R. **Introduction to autonomous mobile robots**. 2nd ed. London: MIT Press, 2004.

WOLF, Denis F. et al. Robótica inteligente: da simulação às aplicações no mundo real. In: Jornadas de Atualização em Informática, 28., 2009, Rio de Janeiro. **Anais...** Rio de Janeiro: PUC, 2009. p. 279-330.

APÊNDICE A – Criação do robô baseado na plataforma LEGO Mindstorms

Antes de iniciar a montagem do robô é necessário separar as peças de acordo com a Figura 21.

Figura 21 - Peças componentes do robô criado



Inicialmente, deve ser montada a roda livre do robô de acordo a respectiva descrição na seção 3.3.2.1. Os passos para montagem da roda podem ser verificados através da Figura 22 à Figura 24.

Figura 22 - Estrutura básica da roda livre

Figura 23 - Acrescentando peças para acoplar ao robô

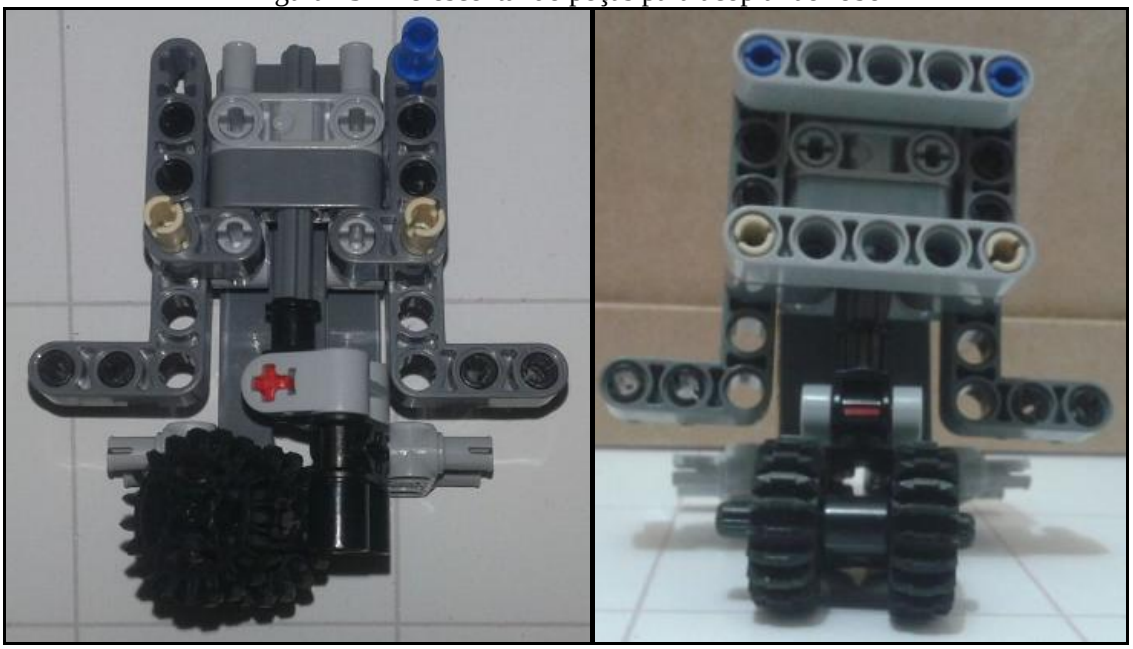
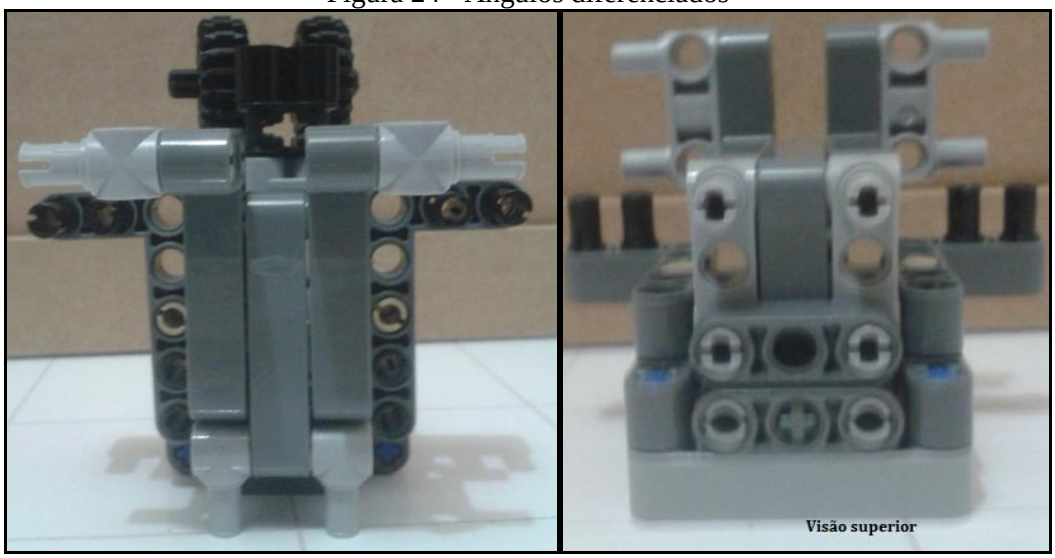


Figura 24 - Ângulos diferenciados



Após finalizar a montagem da roda é necessário conectá-la aos motores do robô. Os detalhes de montagem podem ser verificados na Figura 25 e o resultado na Figura 26.

Figura 25 - Conectar a roda livre aos motores do robô

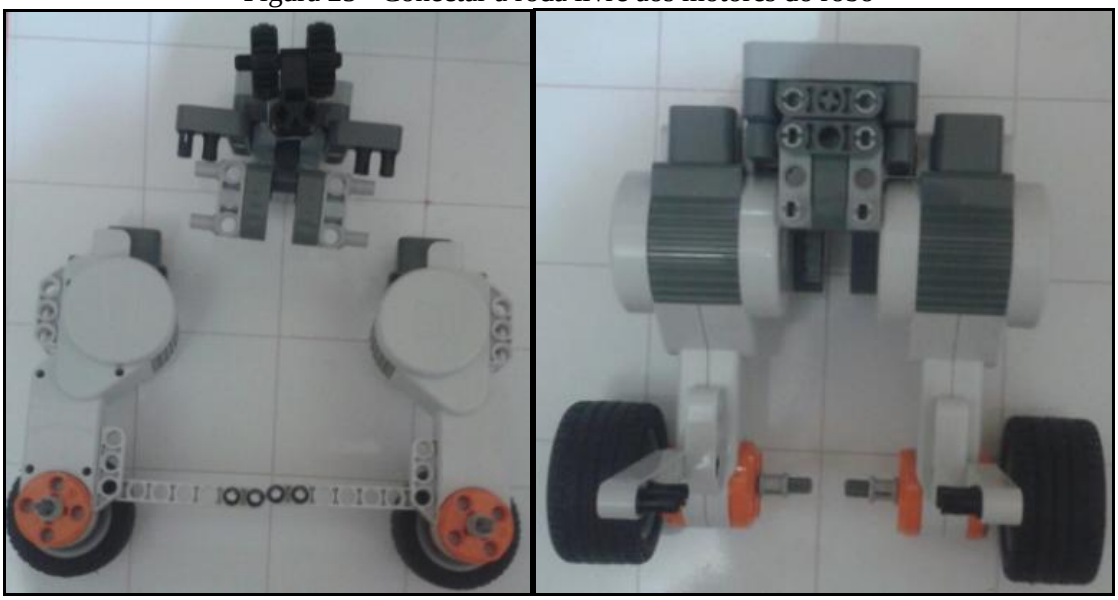


Figura 26 - Robô contendo as duas rodas fixas e a roda livre

Após montar a roda livre, é necessário criar a estrutura que é utilizada como referência para posicionar o robô no cenário. Esta estrutura também é parte da base de apoio do celular e seu detalhamento pode ser acompanhado na Figura 27. Na Figura 28 é possível verificar o resultado desta estrutura acoplada ao que já foi montado do robô e ao bloco programável.

Figura 27 - Estrutura para posicionar o robô no cenário

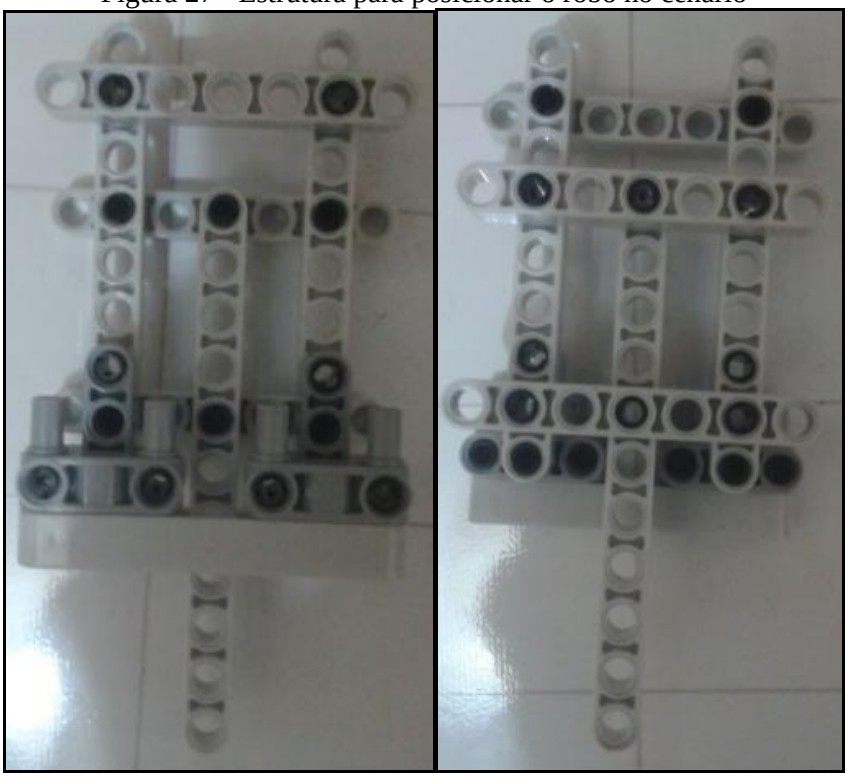
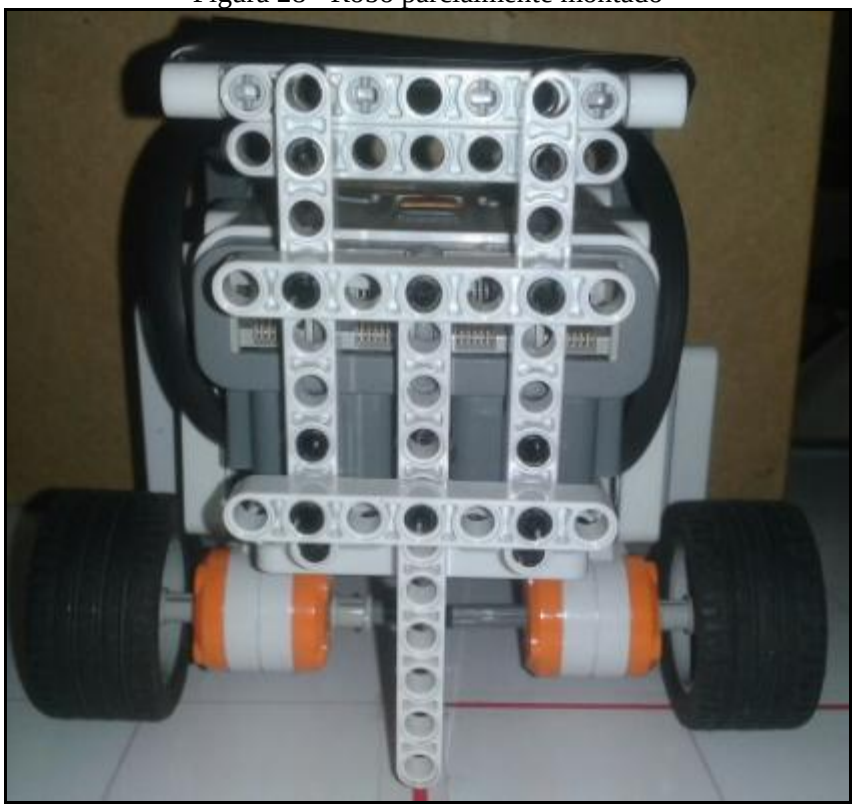


Figura 28 - Robô parcialmente montado



Para que seja possível apoiar o celular acima do robô é necessário criar uma estrutura de acordo com a Figura 29. Acima dela é necessário acoplar a base do celular que é montada conforme Figura 30.

Figura 29 - Estrutura para acoplar base do celular

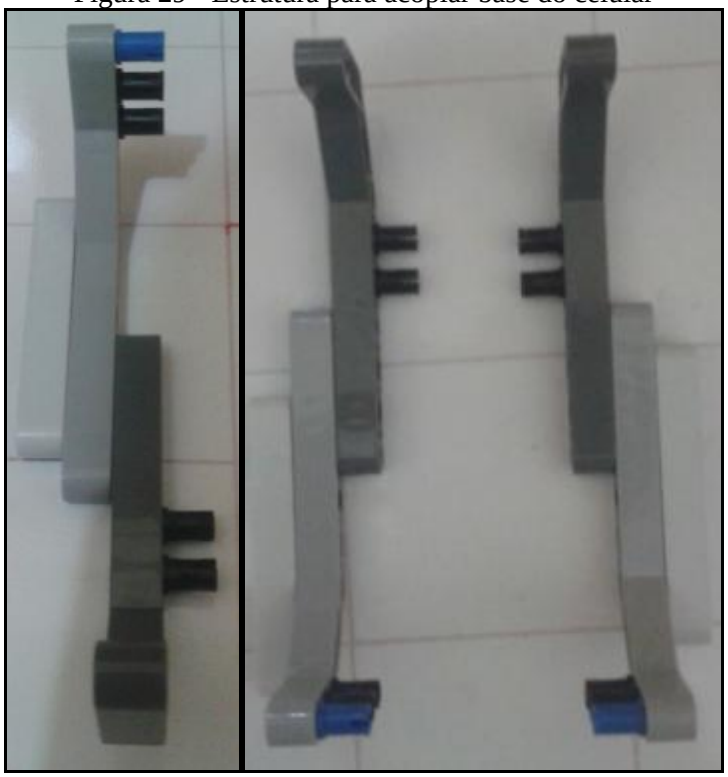


Figura 30 - Base para apoiar o celular

O resultado do desenvolvimento de todas as estruturas descritas resulta no robô utilizado nesse trabalho. Seu formato final fica de acordo com a Figura 31.

Figura 31 - Robô utilizado no presente trabalho