Universidade Federal do Piauí
Campus Senador Helvídio Nunes de Barros
Curso de Bacharelado em Sistemas de Informação

Thiago José Barbosa Lima

PARADINHA: Um robô móvel inteligente

PICOS
2013

Thiago José Barbosa Lima

PARADINHA: Um robô móvel inteligente

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Sistemas de Informação, Campus Senador Helvídio Nunes de Barros da Universidade Federal do Piauí, como parte dos requisitos para obtenção do Grau de Bacharel em Sistemas de Informação, sob orientação de Patricia Medyna Lauritzen de Lucena Drumond e Algeir Prazeres Sampaio.

PICOS

2013

Eu, **Thiago José Barbosa Lima**, abaixo identificado(a) como autor(a), autorizo a biblioteca da Universidade Federal do Piauí a divulgar, gratuitamente, sem ressarcimento de direitos autorais, o texto integral da publicação abaixo discriminada, de minha autoria, em seu site, em formato PDF, para fins de leitura e/ou impressão, a partir da data de hoje.

Picos-PI 23 de setembro de 2013.

Thiago José Barbosa Lima
Assinatura

FICHA CATALOGRÁFICA
Serviço de Processamento Técnico da Universidade Federal do Piauí
Biblioteca José Albano de Macêdo

L732p Lima, Thiago José Barbosa.
Paradinha: um robô móvel inteligente / Thiago José Barbosa Lima. – 2013.
CD-ROM : il. ; 4 ¾ pol. (50 p.)

Monografia(Bacharelado em Sistemas de Informação) – Universidade Federal do Piauí. Picos-PI, 2013.
Orientador(A): Prof. Msc. Patrícia Medyna Lauritzen de L. Drumond

1. Robô Móvel. 2. Arduíno. 3. Inteligência Artificial. I. Título.

CDD 005.1

Thiago José Barbosa Lima

PARADINHA: Um robô móvel inteligente

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Sistemas de Informação, Campus Senador Helvídio Nunes de Barros da Universidade Federal do Piauí, como parte dos requisitos para obtenção do Grau de Bacharel em Sistemas de Informação, sob orientação de Patricia Medyna Lauritzen de Lucena Drumond e Algeir Prazeres Sampaio.

Data de Aprovação: 13/09/13

Prof. MSc. Patricia Medyna Lauritzen de Lucena Drumond _____ UFPI
Prof. MSc. Juliana Oliveira de Carvalho _____ UFPI
Prof. Flávio Henrique Duarte de Araújo _____ UFPI

PICOS
2013

Dedico esta monografia a toda minha família em especial aos meus pais, Teresinha Leal Barbosa, Teodória Barbosa Lima Vidal , Raimundo Barbosa Neto *(In Memorian)* e minha avó Maria Ferreira , por estar sempre ao meu lado em todas as minhas conquistas e decisões.

Agradeço primeiramente a Deus, pelo dom da vida, pela minha capacidade de pensar, ter me proporcionado mais uma vitória e por me guiar nessa jornada de conquista.

Agradeço a toda minha família pela paciência, conselhos, ajuda de custo e por acreditar em meu trabalho, mesmo com as dificuldades encontradas em especial Raimundo Barbosa Neto *(In Memorian)*, Teresa, Teodória, Maria Ferreira, Vivi, Tati, Jardivaldo, Maria, Eligângela, Érica, Leyla, Geraldo, Selma, Tayla, Airton, França, Cassya, Miguel, tios, primos e outros.

Aos meus orientadores Msc. Algeir Sampaio por ter por ter acreditado e confiado no meu potencial, pela oportunidade de trabalhar na área da robótica, e a Msc. Patrícia Medyna por ter dado continuidade ao trabalho, pela paciência e obrigado por ajudar a concluir esse trabalho.

A todos os professores que compõem o curso de Sistemas de Informação, que contribuíram através de seus conhecimentos repassados: Algeir, Arlino, Dennis Sávio, Érik, Flávio Henrique, Francisca Cosme, Frank César, Fredison, Ismael, Ivanildo, Ivenilton, José Ricardo, Janayna, João Santos, Juliana, Júlio César, Leonardo, Patricia Medyna, Patricia Vieira, Rayner, Romuere, Ryan, Tadeu e Cicero.

Aos professores do ensino fundamental e médio, que sem eles não tinha conseguido adiquerir o conhecimento necessário para a conclusão desse trabalho.

Aos servidores técnicos e administrativos pela dedicação e atenção aos alunos: João, Claudiane, Antenor, Manoel Messias, Erasmo.

Aos servidores terceirizados da UFPI pela dedicação: Francisco, Bandeirão, Alan, Justino, seu Bodin, Nice, Manoel Messias, Moaci, Xavier, Djalma e outros.

Aos meus colegas, amigos do curso, da UFPI, em especial a equipe do LIPPO, Allan, Marco Antonio, Wilson, Varton, André, Bruno Rafael, Cliciano, Daniel, Mezzomo, Jonilton, Cavalcante, Ruth, Armando, Abimael, Renan, Klisanderson, Auricélio, Pâmela, Danila, obrigado por estarem presente durante toda essa jornada.

Amigos da turma em especial Melksedek, Rafael Melo, Robson, Itamar, Jailson, Afonso, Rebelato, Gleyson, Israel, Weder, João Mariano, Leonilio, Waldery, Rômulo pelos momentos e aprendizagem que compartilhamos no curso.

A família Lucena que sempre esteve comigo, apoiando e ajudando a crescer moralmente e espiritualmente, em especial Felipe, Cássia, Carol, Felicio, Júnior, Eric, Igor e mais especial a

Francilene e Francisco Lucena (In memorian).

Aos amigos do Roque, Forte, Inhuma, Picos e outros, pela amizade conquistada durante todos esses anos, em especial Antonio José, Romildo, Rafael, Bruno, Ricardo, Reginaldo, Jaelson, Mestre Valdo e João Santos, Raimundão, Leonel, Lucas e demais amigos e colegas.

Ao amigo DePenas, Implumado Geraldo Deris por me ajudar sempre que necessitei, e por ter me motivado sempre em que me encontrava sem animo de continuar a executar as atividade que contribuíram no desenvolvimento dessa pesquisa.

As mocinhas da independência que me ajudaram a superar todos os momentos difíceis e estevem comigo apoiando e ajudando na escrita deste trabalho, em especial Eliene, Lucélia, Dedilsa,Lina Mara, Valéria, Marta.

Nossa maior fraqueza está em desistir. O caminho mais certo de vencer é tentar mais uma vez.

Thomas Edison

É melhor atirar-se à luta em busca de dias melhores, mesmo correndo o risco de perder tudo, do que permanecer estático, como os pobres de espírito, que não lutam, mas também não vencem, que não conhecem a dor da derrota, nem a glória de ressurgir dos escombros. Esses pobres de espírito, ao final de sua jornada na Terra não agradecem a Deus por terem vivido, mas desculpam-se perante Ele, por terem apenas passado pela vida.

Bob Marley

Pouco conhecimento faz com que as pessoas se sintam orgulhosas. Muito conhecimento, que se sintam humildes. É assim que as espigas sem grãos erguem desdenhosamente a cabeça para o Céu, enquanto que as cheias as baixam para a terra, sua mãe.

Leonardo da Vinci

RESUMO

Este trabalho descreve o projeto e a construção de um robô móvel autônomo capaz de seguir o menor caminho em um ambiente real baseado em uma simulação com ou sem obstáculos utilizando o algoritmo A* e uma transmissão de rádio frequência da aplicação no computador para o robô. O robô construído é um protótipo que consiste de 2 motores de passo 5v com dois drives uln2003, uma base chassis TG007, um Arduino Mega 1280, uma mini protoboard, fios, e uma bateria de 7.4 volts utilizada para alimentar o Arduino e demais componentes eletrônicos. O sistema foi testado e os resultados experimentais indicaram que o sistema é eficiente, sendo capaz de se deslocar utilizando o menor caminho entre o ponto de origem e destino sem utilização de sensores.

Palavras-chave: Robô Móvel, Arduino, Inteligência Artificial, Menor Caminho.

ABSTRACT

This paper describes the design and construction of autonomous mobile robot capable to following the shortest path in a real environment based on a simulation with or without obstacles using the A * algorithm and transmission frequency radio of application on the computer to the robot. The constructed robot is a prototype consists of two stepper motor 5v two drives ULN2003, TG007 a base chassis, an Arduino Mega 1280, a mini protoboard, wire, and a battery 7.4 volt used to power the Arduino and remaining electronic components. The system was tested and the experimental results indicate that it is efficient, and is able to move using the shortest path between the origin point and of destination without using sensors.

Keywords: Mobile Robot, Arduino, Artificial Intelligent, Path Shortest.

Lista de Figuras

Lista de Tabelas

Lista de abreviaturas e siglas

A*	A Estrela
API	Application Programming Interface
IA	Inteligência Artificial
IDE	Integrated Drive Eletronics
JDK	Java Development Kit
LED	Light Emitting Diode
PDA	Personal Data Assistent
RF	Rádio Frequência
USB	Universal Serial Bus
XML	eXtensible Markup Language

Sumário

1 Introdução

A Inteligência Artificial surgiu da necessidade de se ter um robô, realizando tarefa semelhante ao homem, e ela vem evoluindo cada vez mais no decorrer dos últimos anos. Os estudos da IA tem se destacado em diversas áreas do conhecimento, e suas aplicações estão presentes no nosso cotidiano. A robótica é uma subárea da IA que vem sendo aplicada em locais como indústrias, campos de pesquisa, operação de risco, exploração de petróleo, entre outros. Atualmente, são desenvolvidos robôs que realizam tarefas de forma eficiente, tarefas antes realizada pelo homem. Máquinas são projetadas para trabalhar de forma escravizada, com o propósito de aumentar o número de produção.

O estudo da robótica móvel é um tema bastante relevante e atual, onde esta área de estudos, pesquisas e desenvolvimento apresentou um grande salto em seu desenvolvimento nas últimas duas décadas. A aplicação prática de robôs móveis junto a diferentes atividades em nossa sociedade vem demonstrando o quão promissor é o futuro desta área (WOLF et al., 2009). Existem situações que necessitam da intervenção de robôs móveis para resolverem problemas de grande risco para o ser humano em locais como o oceano, áreas radioativas e até mesmo em outros planetas. Esses fatores contribuem para que o robô venha substituir o homem em parte, visto que ainda existem muitas tarefas que o robô não consegue substituir o homem. Robôs como esses são desenvolvidos especificamente para realizarem tarefas que envolvem precisão e eficiência.

A minimização de circuitos eletrônicos é uma das maiores contribuições para o avanço da tecnologia atual e projetos robóticos e de automação. No entanto, o desenvolvimento de um projeto de robótica necessita de pesquisas de tecnologias de *hardware* e *software*, utilização de algoritmos de otimização, análise da eficiência de algoritmos, utilização de componentes eletrônicos, tais como, sensores e mecanismos de comunicação. O desenvolvimento do robô começa com a elaboração do projeto, construção do protótipo, aplicação de testes de validação e eficiência.

Diante do exposto, o presente trabalho tem como objetivo principal a descrição do projeto e implementação de um sistema robótico, utilizando simulação de ambientes quadrados, para roteamento de robô móvel capaz de seguir o menor caminho entre um ponto de origem e um ponto de destino definidos com ou sem obstáculos entre eles e avaliar sua eficiência. O percurso que o robô móvel deve seguir não é conhecido previamente, o sistema utiliza um algoritmo de otimização que desenha o menor caminho e transmite as coordenadas para o robô

móvel que o executa.

A metodologia utilizada foi experimental e partiu da construção de um robô com componentes eletrônicos e o uso da placa Arduino para comunicação entre a aplicação desenvolvida no computador e o robô. Um dos problemas principais encontrado no projeto foi à comunicação da aplicação com o robô. A aplicação foi desenvolvida no intuito de executar o algoritmo A-Estrela (A*) e gerar as coordenadas com o melhor caminho, o robô ao receber essas coordenadas é capaz de interpretá-las e atuar no ambiente de forma inteligente.

Este trabalho está organizado em 4 capítulos além da introdução. No Capítulo 2, encontra-se a fundamentação teórica necessária para entendimento do problema e sua aplicação. No capítulo 3 é abordada a concepção do projeto do robô, o algoritmo de resolução do problema do menor caminho e a transmissão das coordenadas. Em seguida, os experimentos computacionais e os resultados obtidos são descritos no capítulo 4. As considerações finais do trabalho apresentadas no capítulo 5.

2 Inteligência Artificial

A Inteligência Artificial (IA) é uma das ciências que vem evoluindo muito ao longo dos anos. Segundo Russell e Norvig (2004), as pesquisas em IA iniciaram após a Segunda Guerra Mundial, e o próprio nome foi cunhado em 1956. É uma ciência que abrange uma enorme variedade de subcampos, tais como, robótica, planejamento autônomo e escalonamento, jogos, diagnóstico, reconhecimento de linguagem e resolução de problemas, dentre outros.

Conforme Fernandes (2005), a IA é a parte da ciência da computação voltada para o desenvolvimento de sistemas de computadores inteligentes, isto é, sistemas que exibem características que estão associadas à inteligência no comportamento humano, como compreensão da linguagem, aprendizado, raciocínio, resolução de problemas, entre outros. Tais características como essas adaptadas em um sistema ou protótipo podem ser consideradas como um agente, uma vez que seja capaz de perceber e atuar em um ambiente.

Um agente é tudo o que pode ser considerado capaz de perceber seu ambiente por meio de sensores e agir sobre esse ambiente. Um agente humano tem olhos, ouvidos e outros órgãos como sensores, e tem mãos, pernas, boca e outras partes do corpo que servem como atuadores. Um agente robótico poderia ter câmeras,sensores ultrassônicos ou infravermelhos, e vários motores como atuadores. Um agente de *software* recebe sequências de teclas dirigidas, conteúdo de arquivos e pacotes de rede como entradas sensoriais e atua sobre o ambiente exibindo algo na tela, gravando arquivos e enviando pacotes. Destacando dois tipos de agentes, o agente reativo simples e o agente inteligente, com características diferentes (RUSSELL; NORVIG, 2004).

O agente reativo simples é capaz de perceber e atuar em um ambiente, por meio de sensores e atuadores. Considerado simples justamente por não ter funcionalidade de aprendizagem. Não é capaz de memorizar estados anteriores, ou seja, ele só percebe e age. Diferenciado do agente reativo simples, o agente inteligente é capaz de perceber, atuar, memorizar e tomar decisões de acordo com o ambiente. Esse agente pode ser *software* ou até mesmo um robô projetado para resolver problemas complexo de forma inteligente.

2.1 Robótica

A robótica é uma área que vem se desenvolvendo bastante nas últimas décadas, por ser um campo de aplicação da IA que integra percepção, raciocínio e ação para a solução dos mais variados problemas encontrados no cotidiano.

Além disso, o estudo da robótica móvel é objeto de pesquisas na atualidade, e sua apli-

cação prática está presente em diferentes atividades em nossa sociedade, demonstrando o quão promissor é o futuro desta área. Por exemplo, seu uso em aplicações domésticas (aspiradores de pó e cortadores de grama robóticos), industriais (transporte automatizado e veículos de carga autônomos), urbanas (transporte público, cadeiras de rodas robotizadas), militares (sistemas de monitoramento aéreo remoto - VANTs, transporte de suprimentos e de armamento em zonas de guerra, sistemas táticos e de combate) e de segurança e defesa civil e militar (controle e patrulhamento de ambientes, resgate e exploração em ambientes hostis), demonstra a grande gama de aplicações atuais dos robôs móveis e os interesses econômicos envolvidos em relação ao seu desenvolvimento e aplicação.

Existem vários tipos de problemas que para sua resolução é necessário o uso da IA e de algoritmos de otimização, tais como, os algoritmos de *Bellman-Ford* e *Dijkstra* que resolvem o problema do caminho mais curto.

2.2 Algoritmo de *Bellman-Ford*

O algoritmo de Bellman-Ford resolve problemas de caminhos mais curto de única origem no caso mais geral, no qual os pesos das arestas podem ser negativos. Dado um grafo orientado ponderado $G = (V, E)$, com origem s, onde V é o conjunto de vértice e E o conjunto de aresta, e função peso $w : E- > R$, o algoritmo de Bellman-Ford retorna um valor booleano indicando se existe ou não um ciclo de peso negativo acessível a partir da origem. Se existir tal ciclo, o algoritmo indica que não existe nenhuma solução. Se não existe tal ciclo, o algoritmo produz os caminhos mais curtos e seus pesos. Como foi citado o algoritmo suporta pesos negativos em suas arestas, onde os outros algoritmos que serão mostrados em seguida não dão suporte a pesos negativos nas arestas (CORMEN et al., 2008).

Segundo Cormen et al. (2008) o algoritmo usa o relaxamento, diminuindo progressivamente uma estimativa $d[v]$ no peso de um caminho mais curto da origem *s* até cada vértice v ϵ V , até alcançar o peso real de caminho mais curto σ (s, v). O algoritmo retorna *TRUE* se e somente se o grafo não conter nenhum ciclo de peso negativo que seja acessível a partir da origem. O (Anexo A) mostra o pseudocódigo do algoritmo de relaxamento e de *Bellman-Ford*, onde o algoritmo de relaxamento é utilizado tanto no o algoritmo de *Bellman-Ford* como no de Dijkstra.

2.3 Algoritmo de *Best-first Search*

O algoritmo de busca *Best-First-Search* usa a função heurística $f(n) = h(n)$ de procura ao nó de destino. Esta procura tentar expandir o nó que é mais próximo ao objetivo, acreditando numa condução rápida ao objetivo. Em vez de selecionar o vértice mais próximo do ponto de

partida, ele seleciona o vértice mais próximo do objetivo. O algoritmo não necessariamente está garantindo encontrar um caminho mais curto. Por usar uma função heurística para guiar o seu caminho em direção ao objetivo muito rapidamente, ele corre muito mais do que o algoritmo de Dijkstra, visando uma maneira mais rápida de se chegar ao objetivo (PATEL, 2013).

De acordo com Galdino e Faria (2009), o algoritmo mantém duas listas. A primeira, aberta, representa o conjunto de nós que ainda podem ser pesquisados e utilizados na composição do caminho. Já a lista fechada, contém todos nós que já foram visitados, e que não precisam ser examinadosnovamente. Com isso o algoritmo mantém em uma lista aberta os nós visitados, evitando eventualmente visitar um nó várias vezes.

2.4 Algoritmo de Dijkstra

O algoritmo de *Dijkstra*, popularmente conhecido por algoritmo do caminho mais curto, é bastante utilizado em projetos de roteamento de redes de computadores, funciona em grafo dirigido ou não dirigido com arestas de peso não negativo, de forma que $f(n) = g(n)$.

O algoritmo de *Dijkstra* resolve o problema do caminho mais curto de única origem em um grafo orientado ponderado $G = (V, E)$ para o caso no qual todos os pesos de arestas não são negativos. Então, suponhamos que $w(u, v) >= 0$ para cada aresta $(u, v)\ \epsilon\ E$. O tempo de execução do algoritmo de Dijkstra é inferior ao do algoritmo de *Bellman-Ford* (CORMEN et al., 2008).

Segundo Ziviani (2007) o algoritmo mantém um conjunto S de vértices cujos caminhos mais curtos até um vértice de origem já são conhecidos. Ao final de sua execução o algoritmo produz uma área de caminhos mais curtos de um vértice origem s para todos os vértices que são alcançados a partir de s. O algoritmo de *Bellman-Ford*, por resolver o problema de caminho mais curto de forma rápida, em um menor tempo, de forma eficiente. Segue no (Anexo A) o pseudocódigo do algoritmo de *Dijkstra* (CORMEN et al., 2008).

2.5 Algoritmo de Busca *A**

O algoritmo *A** é uma técnica de busca utilizada para encontrar o melhor caminho, desde que tenha um ponto de origem e um destino conhecido. O destino é conhecido através de um caminho linear. Esse algoritmo é bastante utilizado em desenvolvimentos de jogos e projetos com robôs. Pode-se utilizar uma função heurística para melhorar a eficiencia da busca pelo menor caminho.

O algoritmo de busca *A** reúne características dos algoritmos de *Best-First Search* e de *Dijkstra*. Segundo Russell e Norvig (2004), a busca *A** é uma forma mais amplamente conhecida da busca pela melhor escolha. Ela avalia nós combinando $g(n)$ (utilizado no algoritmo de

Dijkstra), o custo para alcançar cada nó, e $h(n)$ (utilizado no algoritmo de *Best-FirstSearch*), o custo para ir do nó até o objetivo, baseado na equação $f(n) = g(n) + h(n)$.

Tendo em vista que $g(n)$ fornece o custo do caminho desde o nó n, e que $h(n)$ é o custo estimado do caminho de custo mais baixo desde n até o objetivo. O $f(n)$ receberá o custo estimado da solução de custo mais baixo passando por n. De início, é calculado o custo do nó atual até o nó seguinte, após esse cálculo faz-se a verificação do custo desse nó seguinte até o nó destino. Deste modo, se tentarmos encontrar a solução de custo mais baixo, uma opção razoável será experimentar primeiro o nó com o menor valor de $g(n) + h(n)$. Na verdade, essa estratégia é mais razoável: desde que a função heurística h(n) satisfaça a certas condições, a busca *A** será ao mesmo tempo completa e ótima. Deve-se escolher uma função heurística que mais se adeque ao problema que deseja resolver, com a utilização do algoritmo de busca *A** (RUSSELL; NORVIG, 2004).

Segundo Patel (2013), o algoritmo de busca *A** é a escolha mais popular para encontrar um caminho, porque é bastante flexível e pode ser usado numa grande variedade de contextos, semelhante ao algoritmo *Dijkstra*. É como o algoritmo de *Best-First-Search* em que ele pode usar uma heurística para guiar-se. Com combinação desses dois algoritmos, *A** destaca-se por apresentar ótimos resultados nas resoluções de problemas mais complexos. O (Anexo B) mostra uma representação do pseudocódigo do algoritmo de busca A*, segundo Bourg e Seemann (2004).

2.6 Heurísticas

Segundo Patel (2013), o segredo do sucesso do algoritmo *A** é que ele combina as peças de informação que utiliza o algoritmo de *Dijkstra* (favorecendo os vértices que estão perto do ponto de partida) e de informação que utiliza *Best-First-Search* (favorecendo os vértices que estão perto da meta). Na utilização do algoritmo *A** é importante a boa escolha da heurística, pois a função heurística $h(n)$ diz exatamente uma estimativa do custo mínimo de qualquer vértice até o objetivo. Utilizando-se a heurística serão encontrados caminhos mais curto, mais rapidamente.

Como dito anteriormente, é de grande importância a escolha da função heurística a utilizar no *A**, pois esta implicará no seu tempo de execução da busca. Na próxima seção mostraremos duas heurísticas admissíveis que podem ser usadas no algoritmo de busca *A**.

2.6.1 Distância de *Manhattan*

A distância de *Manhattan* é uma heurística padrão que utiliza uma grade quadrada e são permitidos movimentos em quatro direções: direita, esquerda, para frente e para traz. Em uma

grade quadrada, a heurística deve ser D vezes a distância de Manhattan:

$$h(n) = D * (|node.x - destino.x| + |node.y - destino.y|)$$

Como vimos na fórmula acima, o D representa o menor custo entre os quadrados adjacentes, onde o valor total desse custo é D vezes a soma das diferenças absolutas das coordenadas do nó corrente e o nó destino (PATEL, 2013). A Figura 1 mostra exatamente a representação da distância de *Manhattan*, onde temos um ponto de partida e um destino, em uma grade sem obstáculo.

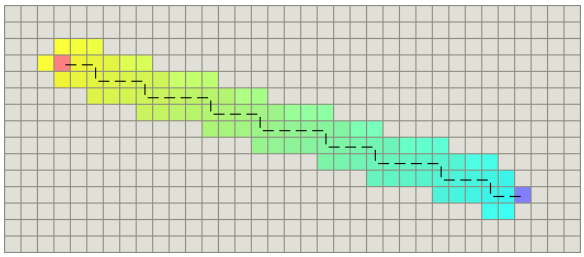

Figura 1 *– Distância de Manhattan*

Fonte: <http://theory.stanford.edu/ amitp/GameProgramming/Heuristics.html>

2.6.2 Distância Euclidiana

Segundo Patel (2013), a distância Euclidiana é utilizada em grade quadrada que permite o movimento em qualquer direção, ou seja, as unidades se movam em qualquer ângulo, em vez de direção de grade, provavelmente usando uma distância em linha reta, mostrada na função abaixo:

$$h(n) = D * \sqrt{(n.x - destino.x)^2 + (n.y - destino.y)^2}$$

.

Desta forma, a distância euclidiana encontrada é menor do que a de Manhattan, ou seja o caminho é mais curto, mas o tempo de execução é mais lento. A Figura 2 mostra com clareza a representação da distância Euclidiana, partindo de um ponto de origem a um destino específico, em uma grade sem obstáculos.

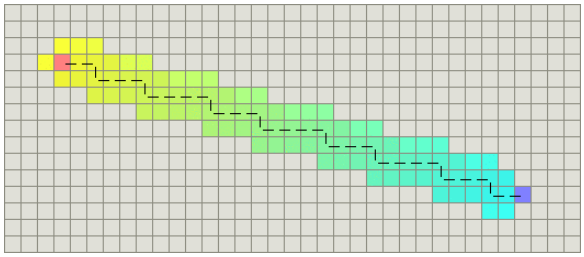

Figura 2 *– Distância de Euclidiana*

Fonte: <http://theory.stanford.edu/ amitp/GameProgramming/Heuristics.html>

3 Um robô móvel inteligente

O projeto apresenta a utilização do algoritmo *A** aplicado a um robô móvel terrestre. A concepção do robô móvel foi dividida em três etapas: a aplicação no computador; a transmissão das coordenadas da aplicação para o robô e; a execução em um ambiente real. A aplicação desenvolvida é responsável pela execução do algoritmo *A**, que a partir de sua execução são geradas as coordenadas com o menor caminho, e transmitidas via comunicação serial, para um Arduino conectado à aplicação. A transmissão dessas coordenadas para o robô acontece através de um Arduino conectado ao computador que recebe as coordenadas e é responsável por transmiti-las para o robô, usando um transmissor RF. A terceira etapa acontece quando o robô recebe essas coordenadas, interpreta e se desloca de uma origem a um destino, percorrendo o menor caminho em um ambiente real, similar ao ambiente virtual definido na aplicação.

A aplicação foi escrita na linguagem Java, utilizando como plataforma a IDE *NetBeans*, juntamente com o JDK e a biblioteca RXTX. Com uma interface amigável e intuitiva, se torna uma aplicação de fácil entendimento, sendo muito útil para quem deseja trabalhar com projetos envolvendo Arduino, IA, algoritmos de otimização como o *A**, entre outros.

O *A** foi implementado na linguagem *Java*, com a finalidade de definir ambientes com ou sem obstáculos. O propósito principal deste algoritmo é buscar pelo menor caminho possível em um ambiente pré-definido, partindo de uma origem até um destino específico, com o auxílio de uma heurística para melhorar a eficiência em sua busca. A heurística utilizada foi a de *Manhattan*, pelo fato de poder se movimentar em quatro direções, sendo a que mais se adequou ao projeto. Com isso o robô pode locomover-se em quatro direções possíveis em um ambiente real baseado no virtual, como pré-requisito ambos os ambientes devem ser iguais.

3.1 Infraestrutura para o desenvolvimento do protótipo

Para o desenvolvimento do protótipo foi necessário o estudo e utilização de ferramentas específica para desenvolver todo o trabalho. Os estudos realizados foram com base em pesquisas e projetos relacionados à área da robótica, com ênfase na utilização de Arduino e IA.

3.2 Linguagem de programação *Java*

Segundo Deitel e Deitel (2006), o *Java* foi anunciado formalmente pela *Sun*, em uma conferência que aconteceu em maio de 1995. Chamou a atenção da comunidade de negócios por causa do enorme interesse na *World Wide Web*. Não é uma linguagem muito antiga, mas já se expandiu no mundo inteiro.

A linguagem *Java* é conhecida mundialmente. É uma linguagem orientada a objetos fazendo parte da quinta geração juntamente com *C++, Delphi, Visual Basic, C#, VB.Net*. O *Java* fornece uma variedade de bibliotecas pré-definidas que auxiliam bastante no desenvolvimento de aplicações. Destaca-se das outras linguagens orientadas a objeto por dar suporte a vários tipos de aplicações, com propósitos diferentes.

Java é a base para praticamente todos os tipos de aplicação em rede e é o padrão global para o desenvolvimento e entrega de aplicações móveis, jogos, conteúdo baseado na *web* e *software* da empresa. Com mais de 9 milhões de desenvolvedores em todo o mundo, *Java* permite desenvolver e implementar de forma eficiente aplicativos interessantes e serviços. Com ferramentas abrangentes, um ecossistema maduro, e desempenho robusto, *Java* oferece portabilidade aplicações em ambientes de computação. (ORACLE, 2013).

Para desenvolvimento de aplicações na linguagem Java, é necessário apenas o *kit* de desenvolvimento *Java development* (*JDK*) e uma plataforma (*IDE* de desenvolvimento). Sendo quepara que as aplicações escritas na linguagem Java sejam executadas, deve-se ter o *JDK* instalado no computador.

3.3 Plataforma *Netbeans*

A Plataforma *NetBeans* é uma estrutura genérica para aplicações *Swing*. Ele fornece o encanamento que, antes, cada desenvolvedor tinha que conectar ações para itens de menu, barra de ferramentas e atalhos de teclado, gerenciamento de janelas, e assim por diante (NETBEANS, 2013).

O *NetBeans* apresenta uma interface de fácil manipulação, é uma plataforma que funciona tanto em sistemas operacionais *Windows* como *Linux*, facilitando assim para os desenvolvedores, que podem optar em qual sistema operacional será desenvolvido suas aplicações.

A Plataforma *NetBeans* fornece uma arquitetura de aplicativo confiável e flexível. O aplicativo não tem que olhar qualquer coisa como uma IDE . Ele pode te salvar anos de tempo de desenvolvimento. Como a arquitetura da plataforma *NetBeans* é modular, é fácil criar aplicações que são robustas e extensíveis. Uma arquitetura que incentiva práticas de desenvolvimento

sustentável (NETBEANS, 2013).

A Plataforma NetBeans fornece uma infraestrutura para registrar e recuperar implementações de serviço, o que permite minimizar as dependências diretas entre os módulos individuais e que permitem uma arquitetura flexível. Também fornece um sistema de arquivos virtual. Inclui uma *Application Programming Interface* (API) unificada proporcionar fluxo orientado acesso a estruturas planas e hierárquicas, como baseados em disco arquivo sem servidores locais ou remotos, de memória baseados em arquivos e documentos *eXtensible Markup Language (XML)* (NETBEANS, 2013).

3.4 Comunicação serial

Na troca de dados entre dois sistemas, podem ser usados dois tipos de comunicação, via serial ou via paralela. Na comunicação via serial, os dados são enviados de cada vez, já na paralela os dados são enviados de forma simultanea. Os projetistas da linguagem Java deixaram de lado o desenvolvimentos das bibliotecas para comunicação serial.

Segundo Devmedia (2013), o Java possibilita que um mesmo *software* possa ser executado em diversas plataformas sob uma compilação, não sendo necessário reescrever ou recompilar o *software* para que esteja disponível em outras plataformas, seja ela, *Windows*, *Linux*, *Mac* ou *PDA's*. Com o Java se torna mais complexo a tarefa de chamar *API's* nativas dos sistemas operacionais ou ainda fazer comunicação diretamente com o *hardware*. Para resolver esse problema a *Sun* e demais empresas envolvidas no desenvolvimento Java disponibilizaram diversas *API's* para facilitar o trabalho.

A *API RXTX* da suporte para a comunicação tanto serial quanto paralela, ela também possibilita a comunicação via *USB* com a mesma *API*. É baseada na *API Javacomm* distribuída pela própria *Sun*, com a vantagem sobre esta de que ela é portátil para *Windows*, *Linux* e *Mac*, enquanto que a Javacomm em sua última versão só é portátil para *Linux* (DEVMEDIA, 2013).

3.5 Arduino

Arduino é uma plataforma *open-source* de prototipagem eletrônica baseada em *hardware* e *software*. É destinado a artistas, designers, *hobbyists*, e qualquer pessoa interessada em criar objetos ou ambientes interativos (ARDUINO, 2013).

A placa Arduino é composta por entradas e saída analógica e digital, sendo que nessas entradas podem ser conectados componentes eletrônicos para receber informações e para executar ações a partir das informações obtidas. Com a placa é possível desenvolver projetos simples (fazer um *led* piscar) e projetos mais complexo (automação de uma residência).

Arduino pode sentir o ambiente através da recepção de entrada a partir de uma variedade

de sensores e pode afetar os seus arredores por luzes, controladores, motores e outros atuadores. O micro controlador na placa é programado usando a linguagem de programação do *Arduino* (baseado em *Wiring*) e o ambiente de desenvolvimento *Arduino* (baseada em processamento). Os Projetos com Arduino pode ser de forma autônomo ou eles podem se comunicar com *software* rodando em um computador (por exemplo, *Flash*, *Processing*, *MaxMSP*). A programação do *Arduino* é baseada em *C/C++*.Por ser um *hardware* livre, é disponibilizado uma IDE para o desenvolvimento de aplicações, constituindo-se de uma linguagem própria (ARDUINO, 2013).

As placas podem ser construídas à mão ou compradas e o software pode ser baixado gratuitamente. Os projetos de *hardware* de referência estão disponíveis sob uma licença de código aberto, você é livre para adaptá-los às suas necessidades (ARDUINO, 2013).

3.6 Transmissão rádio frequência

A transmissão sem fio é muito importante quando se deseja transmitir informações de uma base a outra em longa distância. A escolha do tipo de transmissão a ser utilizada é mais importante ainda. Um dos tipos bem conhecido é a transmissão por ondas de rádios.

As ondas de rádio são fáceis de gerar, podem percorrer longas distâncias e penetrar facilmente nos prédios; portanto, são amplamente utilizadas para comunicação, seja em ambientes fechados ou abertos. As ondas de rádio também são omnidirecionais, o que significa que elas viajam em todas as direções a partir da origem. Um fator importante é que o transmissor e o receptor não precisam estar alinhados fisicamente (TANENBAUM; WETHERALL, 2011).

Segundo Tanenbaum e Wetherall (2011), as propriedades das ondas de rádio dependem da frequência a ser utilizada. Em baixas frequências, as ondas de rádio atravessam bem os obstáculos, mas a potência cai abruptamente à medida que a distância da origem aumenta – pelo menos cerca de 1/ r2 no ar –, pois a energia do sinal se espalha de forma mais estreita por uma superfície maior. Essa atenuação é chamada de perda no caminho. Em altas frequências, as ondas de rádio tendem a viajar em linha reta e a ricochetear nos obstáculos. Existem vários fatores que podem gerar perdas como conectores, cabos, entre outros. Em outras palavras, em todas as frequências seja ela alta ou baixa, as ondas de rádios podem sofrer interferência causadas por equipamentos elétricos.

3.7 O Protótipo

O protótipo do robô móvel terrestre do tipo carrinho, denominado Paradinha, cuja foto é mostrada na Figura 3, foi construído utilizando dois motores de passos 5v com dois *drives* uln2003 , uma base *chassis* TG007, um Arduino Mega 1280, uma mini *protoboard*, fios, e uma

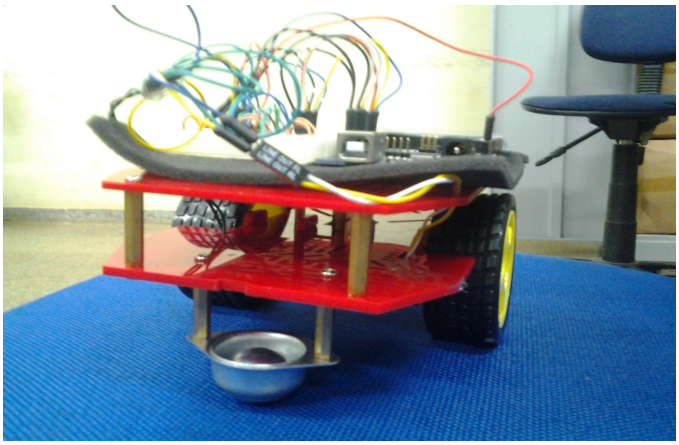

Figura 3 *– Robô Móvel Paradinha*

bateria de 7.4 *volts* utilizada para alimentar o Arduino e demais componentes eletrônicos. O robô foi construído com a finalidade de se deslocar em ambiente de forma autônoma, com as coordenadas do menor caminho desse ambiente, partindo de um ponto de origem até um ponto de destino.

O Paradinha foi desenvolvido para atuar em um ambiente real, com e sem obstáculos, baseado em um plano cartesiano. Ele inicia com uma posição atual qualquer e a cada iteração da execução de uma coordenada, essa posição pode ou não ser alterada. Essas posições são representadas como D, E, C, e B, direita, esquerda, acima, e abaixo, respectivamente.

A execução do menor caminho em um ambiente real é feita a partir do momento em que o robô recebe a transmissão do vetor contendo as coordenadas do percurso, e é definida que a posição atual do robô é a direita **pA = 'D'**. Em seguida, ele executa a função **executaCaminho**(). A função **executaCaminho**() é responsável por percorrer todo o vetor de coordenadas; a cada iteração verifica qual a próxima posição o robô deve seguir e executar o caminho de cada coordenada. Foi criado um laço de repetição com a variável **i** recebendo o valor 2 inicialmente e sendo incrementada de dois até percorrer todo o vetor com as coordenadas. As posições 0 e 1 do vetor de coordenadas, corresponde a posição inicial que o robô se encontra. A Tabela 1 mostra como essas posições são atualizadas a cada iteração.

CONDIÇÃO	POSIÇÃO	PRÓXIMA POSIÇÃO
if(x < vetor[i])	p = 'D'	Direita
if(x > vetor[i])	p = 'E'	Esquerda
if(y < vetor[i + 1])	p = 'C'	Acima
if(y > vetor[i + 1])	p = 'B'	Abaixo

***Tabela 1** – Verifica Posição*

Ao final de cada iteração é executada função **verificaCaminho(p)** que é responsável por girar o robô da posição atual **pA** para posição seguinte **p**, passada por parâmetro. A Tabela 2 mostra todas as possibilidades para que essa ação de movimento do robô aconteça de forma correta. Quando finaliza a execução da função **verificaCaminho(p)**, a posição atual **pA** é atualizada com o valor da posição seguinte **p**, o x recebe o valor do **vetor[i]**, **y** recebe o valor do **vetor[i + 1]**, e em seguida antes da próxima iteração o robô segue em frente, com a execução do comando **motor.step(LOC, 2)**.

CONDIÇÃO	ATIVAÇÃO	EXECUÇÃO
if(pA == 'D' && p == 'C')	motor.step(DIR, 4)	Gira uma vez para esquerda
if(pA == 'D' && p == 'B')	motor.step(DIR, 6)	Gira uma vez para direita
if(pA == 'D' && p == 'E')	motor.step(DIR, 4)	Gira duas vez para esquerda
if(pA == 'E' && p == 'D')	motor.step(DIR, 4)	Gira duas vez para esquerda
if(pA == 'E' && p == 'C')	motor.step(DIR, 6)	Gira uma vez para direita
if(pA == 'E' && p == 'B')	motor.step(DIR, 4)	Gira uma vez para esquerda
if(pA == 'C' && p == 'D')	motor.step(DIR, 6)	Gira uma vez para direita
if(pA == 'C' && p == 'B')	motor.step(DIR, 6)	Gira duas vez para direita
if(pA == 'C' && p == 'E')	motor.step(DIR, 4)	Gira uma vez para esquerda
if(pA == 'B' && p == 'D')	motor.step(DIR, 4)	Gira uma vez para esquerda
if(pA == 'B' && p == 'C')	motor.step(DIR, 6)	Gira uma vez para direita
if(pA == 'B' && p == 'E')	motor.step(DIR, 6)	Gira uma vez para direita

***Tabela 2** – Coordenadas de movimento do robô*

3.8 A Aplicação

Ao executar a aplicaçao, é exibida uma tela principal, como mostra a Figura 4, dividida em três áreas: central, lateral esquerda e lateral direita. Na área central, encontra-se o

logotipo do Laboratório de Investigações e Pesquisas em Poéticas Digitais (LIPPO), onde foi desenvolvido o projeto.

Figura 4 *– Tela Principal*

Na lateral esquerda são encontradas todas as opções necessárias para que a aplicação seja executada de forma correta, é composta por três divisões. A primeira divisão está relacionada à conexão, contendo cinco opções: portas ativas, atualizar portas, conectar, desconectar e limpar monitor. A segunda da barra lateral esquerda é a execução, onde temos o $A*$ sem obstáculos, $A*$ com obstáculos, demonstração do $A*$, e demonstração em um ambiente, baseado em um ambiente real. Na terceira divisão da barra lateral esquerda encontra-se as opções de controlar o robô, ajuda e sair da aplicação. Cada uma dessas opções citadas acima podem ser acionadas com apenas um clique. Na lateral direita encontra-se um monitor serial, usado para mostrar informações do funcionamento da aplicação. Na parte inferior da tela principal, apresenta uma barra de status.

Na opção de **portas ativas** da tela principal, são exibidas as portas seriais ativas, ou seja, todos os dispositivos conectados ao computador, como Arduino. Caso o Arduino seja conectado antes de iniciar a aplicação ele será listado normalmente, caso contrário deverá utilizar o botão ao lado das portas ativas, para que um novo dispositivo seja listado e selecionado. Com o dispositivo selecionado poderá estabelecer uma conexão com o mesmo, clicando no botão **Conectar** da tela principal. A partir do momento que estabelecer uma conexão, o usuário poderá desconectar o dispositivo normalmente. No caso de tentar clicar em **Conectar** e não tiver selecionado nenhuma porta, será exibida uma mensagem de erro, como mostra na Figura 5 abaixo. Esse procedimento é feito justamente para que a aplicação funcione corretamente, mesmo para

usuários não familiarizados com a mesma.

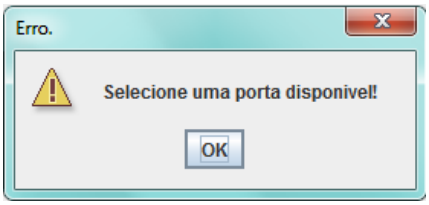

***Figura 5** – Mensagem de Erro na conexão de porta*

O evento para listar todas as portas seriais, é obtido a partir do código mostrado abaixo da classe **SerialComm**, definido um vetor de *String* **listaPortas** que é instanciada com a quantidade de portas ativas. Após a instância é criada uma variável do tipo **Enumeration** que recebe todos os elementos referentes às portas ativas, enquanto existir elementos **e.hasMoreElements()**, será chamado o próximo **e.nextElement()** e adicionará uma nova porta **cpi.getName()** no vetor de *String* **listaPortas[i]** na posição de **i** que inicia com 0 e que a cada iteração a variável **i** será incrementada de 1.

```
public void addPortas(){
    listaPortas = new String[qtdPortas()];
    Enumeration e = CommPortIdentifier.getPortIdentifiers();
    int i = 0;
    while(e.hasMoreElements()){
        CommPortIdentifier cpi = (CommPortIdentifier)e.nextElement();
        listaPortas[i] = cpi.getName();
        i++;
    }
}
```

O método **execute(StringportName)** da classe **SerialComm**, é responsável por estabelecer a conexão serial a partir de um porta selecionada. É uma função booleana que retorna verdadeiro quando a conexão é estabelecida com sucesso e retorna falso quando não se estabeleceu uma conexão. Isso pode acontecer caso a porta esteja ocupada. Recebendo a porta selecionada por parâmetro, a função irá estabelecer a conexão com o Arduino conectado a aplicação. Deve-se ter cuidado é na configuração da porta, que são os seguintes parâmetros **serialPort.setSerialPortParams(9600, SerialPort.DATABITS_8, SerialPort.STOPBITS_1, SerialPort.PARITY_NONE)**. Essa configuração deve ser igual a do Arduino, pois caso contrário, não será possível estabelecer uma comunicação direta. O atributo **inputStream** receberá

serialPort.getInputStream(), onde permitirá que a aplicação receba dados vindo do Arduino, e de forma análoga, o atributo outputStream receberá **serialPort.getOutputStream()**, fazendo com que a aplicação possa transmitir dados para o Arduino. Esses dados a serem transmitidos por comunicação serial podem ser do tipo inteiro ou byte. Ambos os tipos utilizam o mesmo método para realizar a transmissão, **outputStream.write(dados)**. No caso da transmissão de bytes pode ser transmitido um vetor de bytes, diferenciando da transmissão de inteiros.

```
public boolean execute(String portName) {
    CommPortIdentifier portId = getPortIdentifier(portName);
    if(portId != null) {
        try {
            serialPort = (SerialPort) portId.open(
            this.getClass().getName(), 2000);
            inputStream = serialPort.getInputStream();
            outputStream = serialPort.getOutputStream();
            serialPort.addEventListener(this);
            serialPort.notifyOnDataAvailable(true);
            serialPort.setSerialPortParams(
               9600,
               SerialPort.DATABITS_8,
               SerialPort.STOPBITS_1,
               SerialPort.PARITY_NONE);
            return true;
        }
        catch (PortInUseException e) {}
        catch (IOException e) {}
        catch (UnsupportedCommOperationException e) {
            e.printStackTrace();
        }
        catch (TooManyListenersException e) {}
    }
    }
    return false;
}
```

No monitor serial são exibidos todos os dados vindo do Arduino, a partir do evento

SerialPortEvent.DATA_AVAILABLE da classe **SerialComm**, pois quando esse evento estiver habilitado, será chamado o método **recebeSerial()**. Essa função é responsável por receber os *bytes* via comunicação serial, sendo convertidos em *String* e adicionados ao monitor serial localizado na tela principal. Os dados são mostrados no monitor de forma sequencial.

3.9 Algoritmo *A**

No método **pesquisar()**, inicia-se procurando um quadrado com o menor custo em **F** na lista aberta. Para que possa ser feita essa procura até encontrar o quadrado menor de custo **F**, foi declarada uma variável local Matriz **corrente = listaAberta.get(0)**, que recebe o primeiro elemento da lista aberta (listaAberta.get(0)). É usado um laço de repetição que inicia a variável **i** com o valor 1 e incrementado de 1 até que o **i** seja igual ao número de elementos da lista aberta **i < listaAberta.size()**. A cada iteração é verificado se o valor do custo de **F** do quadrado corrente é maior do que o valor do custo **F** da lista aberta na posição de **i**, **corrente.getCustoF() > listaAberta.get(i).getCustoF()**, sendo verdadeiro esse teste, o valor corrente será substituído pelo valor do custo da lista aberta na posição de i, **corrente = listaAberta.get(i)**. Esse teste acontece a cada iteração, onde a principal finalidade é encontrar na lista aberta um quadrado com o menor custo de **F** para ser adicionado na variável corrente, como mostra no trecho abaixo.

```
Matriz corrente = listaAberta.get(0);
 for (int i = 1; i < listaAberta.size(); i++) {
   if (corrente.getCustoF() > listaAberta.get(i).getCustoF()){
     corrente = listaAberta.get(i);
 }
}
```

Após encontrar o quadrado com o menor **custo F** na lista aberta, esse quadrado será adicionado na lista fechada **listaFechada.add(corrente)**, removido da lista aberta **listaAberta.remove(corrente)** e adicionado na lista tudo **listaTudo.add(corrente)**. Essa **listaTudo** guarda todos os quadrados correntes pesquisado em toda execução deste método recursivo. Em seguida, é feito um teste para saber se o quadrado corrente é igual ao quadrado destino **corrente == destino**, caso positivo, será retornado verdadeiro e assim a busca foi concluída, caso contrário a busca pelo menor caminho continua. Após esse teste, é feita uma varredura a todos os quadrados adjacentes ao corrente, ou seja, direita, esquerda, acima e abaixo, da seguinte forma, é declarada duas variáveis **x** e **y**, que recebem, respectivamente, o valor de **corrente.getX()** e **corrente.getY()**. Em seguida são declaradas quatro variáveis inteiras: **direita**, **esquerda**,

acima, **abaixo**. A variável direita receberá o valor de x + 1 quadrado adjacente da direita, a esquerda receberá o valor de x – 1, quadrado adjacente da esquerda, a acima será atribuída o valor de y – 1, quadrado adjacente acima, e a abaixo receberá o valor de y + 1, quadrado adjacente abaixo. Todas essas posições citadas acima são em relação ao quadrado corrente, que de certa forma é baseado em um plano cartesiano. Quando for alterado o eixo de x, temos duas possibilidades de movimentos tanto para direita quanto para esquerda. Caso seja alterado o y teremos também duas possibilidades de movimentos, para cima e para baixo.

Após ser executado o que foi citado no parágrafo anterior, são realizados quatro testes, e em cada teste pode ou não ser atualizados os valores dos custos de **G** e **H** e podendo ser ou não adicionado o nó corrente à lista aberta. Para que isso acontece, deve-se atender aos seguintes testes, direita menor que o tamanho da matriz, esquerda maior ou igual a 0, acima maior ou igual a 0, e abaixo menor que o tamanho da matriz. O código abaixo mostra um teste feito no quadrado adjacente da direita do quadrado corrente. Em cada teste é declarado um objeto **Matriz adjacenteDireta = grade[direita][y]**, esse o objeto **adjacenteDireta** recebe uma coordenada **grade[direita][y]**, substituíndo o valor de da coordenada x por o valor da vaiável da direita e permanecendo o valor da coordenada y.

```
if (direita < grade.length) {
   Matriz adjacenteDireta = grade[direita][y];
   if (!listaFechada.contains(adjacenteDireta) &&
       !listaBloqueios.contains(adjacenteDireta)) {
       int custoG = corrente.getCustoG() + 1;
       int custoH = Math.abs(
              destino.getX() - adjacenteDireta.getX()) +
              Math.abs(destino.getY() - adjacenteDireta.getY()
       );

       if (!listaAberta.contains(adjacenteDireta)) {
           adjacenteDireta.setPai(corrente);
           listaAberta.add(adjacenteDireta);
           adjacenteDireta.setCustoG(custoG);
           adjacenteDireta.setCustoH(custoH);
      } else {
           if (adjacenteDireta.getCustoH() > custoH) {
                adjacenteDireta.setPai(corrente);
                adjacenteDireta.setCustoG(custoG);
```

```
                    adjacenteDireta.setCustoH(custoH);
                }
            }
        }
    }
```

Com base no código acima, após declarar o objeto adjacente os custos de G e H são calculados, com a condição de que o objeto adjacente declarado no início não pertença à lista fechada e nem a lista de bloqueios. O custo G é calculado a partir do custo do nó corrente somando mais 1. O custo H é calculado de forma Heurística. Caso o valor adjacente atual não pertença à lista aberta o nó corrente será adicionado à lista aberta e os custos G e H serão atualizados. Caso o nó adjacente já estiver na lista aberta, é verificado se o custo anterior de H é maior que o custo atual de H, e se o custo anterior for maior, os custos de G e H são atualizados, respectivamente. Ao fim deste teste, são verificados os testes restantes, que particularmente não mudam muito em sua execução. Neste caso o que muda são apenas os objetos a serem declarados.

O último teste do método pesquisar é para verificar se a lista aberta é igual a zero, pois se não contiver nenhum elemento na lista aberta, significa que não existe caminho válido. Esse método pesquisar é recursivo e só termina quando encontra um menor caminho válido ou quando não existe nenhum caminho, sendo que após essa execução é adicionado o menor caminho a lista de caminho.

3.10 Interface do *A**

A interface gráfica foi implementada para demonstrar visualmente a execução do algoritmo *A**, e desenvolvida utilizando apenas um *JFrame*, um *JPanel* e uma matriz quadrada de *JButton*. Dentro do *JFrame* foi adicionado um *JPanel* e de acordo com o tamanho da matriz definida, os **JButton** são ajustados e adicionados normalmente no *JPanel*. Cada *JButton* representa um quadrado ou célula da matriz, sempre levando em consideração a representação de um plano cartesiano. A Figura 6 apresenta uma representação gráfica da execução do algoritmo *A** sem obstáculos. O tamanho da matriz gerada pelo algoritmo é igual ao da interface, onde foi definida uma matriz quadrada 10 x 10, sendo um ambiente sem obstáculos. O ponto de partida é representado pela cor verde, que inicia na posição com coordenada (0,0), o caminho é representado pela cor amarela e o ponto vermelho é o destino na posição com coordenada (9,9). Essa representação é de fácil entendimento, demonstra que o algoritmo funciona corretamente na busca pelo menor caminho, sendo que o mesmo seleciona apenas o menor caminho possível entre as coordenadas definidas.

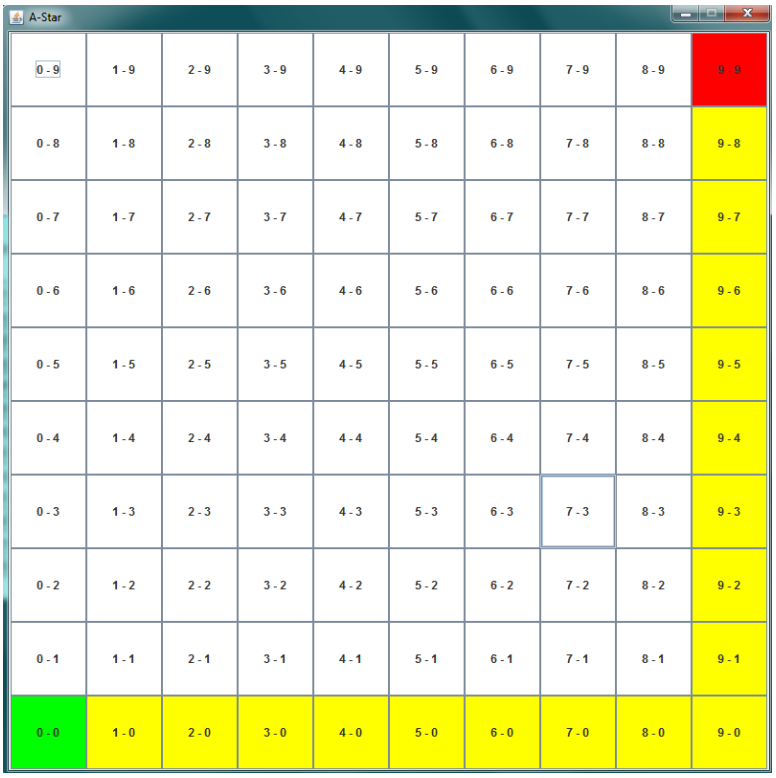

Figura 6 *– Interface do A* sem obstáculos*

O algoritmo *A** é adaptativo a ambientes com e sem obstáculos. Em um ambiente sem obstáculos sempre existe um caminho. No ambiente com obstáculos, pode existir casos que não é possível encontrar um caminho. Se não existir um caminho na execução do algoritmo *A** em ambiente com obstáculos, a aplicação retorna uma mensagem de erro na Figura 7, indicando que o caminho não foi encontrado.

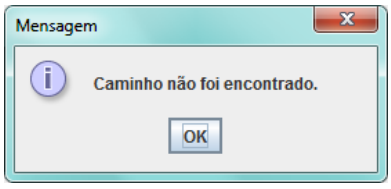

Figura 7 *– Mensagem de caminho não encontrado*

A Figura 8 apresenta um exemplo de um ambiente com obstáculos com a impossibilidade de caminho entre a origem e o destino, onde os obstáculos representados pelas células de cor preta, a origem de cor verde e o destino de cor vermelha.

Quando existir caminho válido em um ambiente com obstáculos, o algoritmo *A** seleciona o menor caminho desviando os obstáculos. A Figura 9 apresenta um exemplo de um caminho válido entre o ponto de origem e o ponto de destino em um ambiente com obstáculos, o ponto de origem é representado pela cor verde, o ponto de destino pela cor vermelha, os

obstáculos pela cor preta e o caminho válido pela cor amarela.

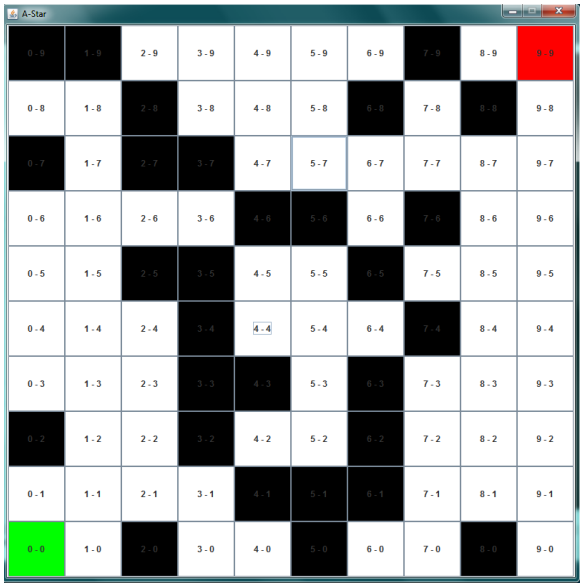

Figura 8 *– Exemplo de Interface do A* com obstáculos sem caminho*

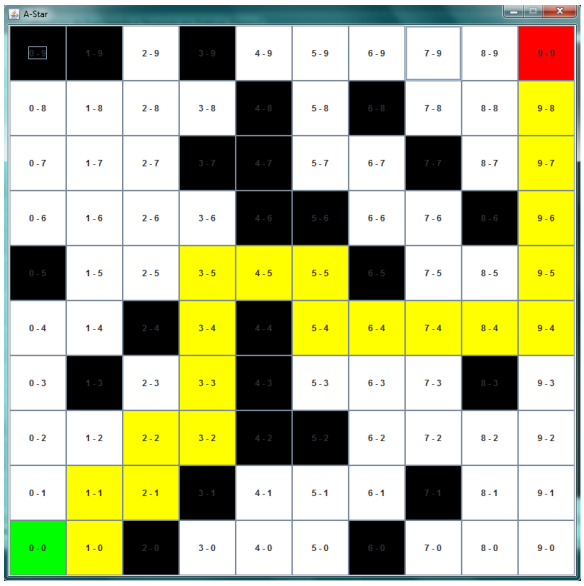

Figura 9 *– Exemplo de um caminho válido para um ambiente com obstáculo*

A implementação dessas interfaces foi baseada em uma matriz, uma demonstração para um melhor entendimento da execução do *A** está localizado no evento do botão **Demo A***. Essa opção de demonstração traz como recurso, a partir do menu **A-Star** (Figura 10), a escolha do tamanho da matriz, a definição de quantos obstáculos dever conter, opção de gerar obstáculos aleatoriamente, procurar o menor caminho, e reenviar o caminho. A Figura 10 mostra a tela inicial após clicar no botão **Demo A***, com as opções da aplicação que devem ser utilizadas pelo usuário.

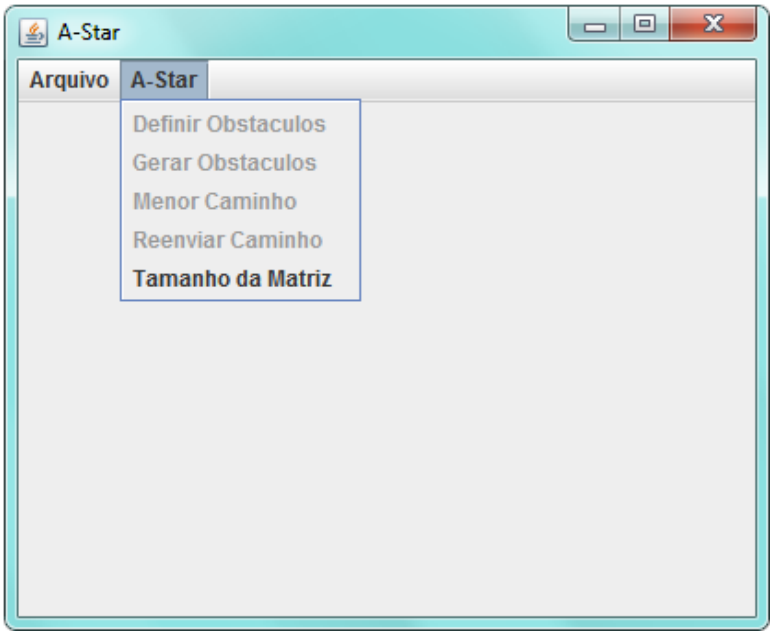

Figura 10 *– Tela Inicial do A-Star*

Para melhorar o entendimento do usuário foram implementadas todas essas funções que é mostrada no menu da tela inicial de A-Star (Figura 10). Ao clicar na opção Tamanha da Matriz, é exibida uma janela para que possa ser inserida na mesma o **tamanho da matriz** quadrada, que pode variar de 3 a 80 e confirmar em ok. Essa janela de definição do tamanho da matriz é apresentada na Figura 11.

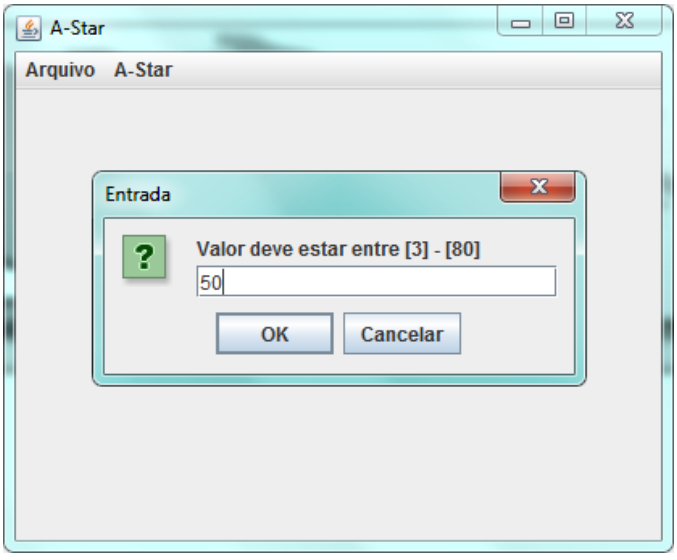

Figura 11 *– Definição do Tamanho da Matriz*

Na Figura 12 é mostrada uma matriz formada por um conjunto de *JButtons*. Após a definição do tamanho da matriz, é definida a quantidade de obstáculos que irá conter no ambiente quadrado. Essa definição de quantidade de obstáculos pode variar de 0 até o tamanho

da matriz. Os obstáculos são adicionados de acordo com o valor que o usuário informar, onde o usuário pode optar por definir os obstáculos ou gerá-los aleatoriamente. No exemplo (Figura 12) o usuário definiu 3 obstáculos.

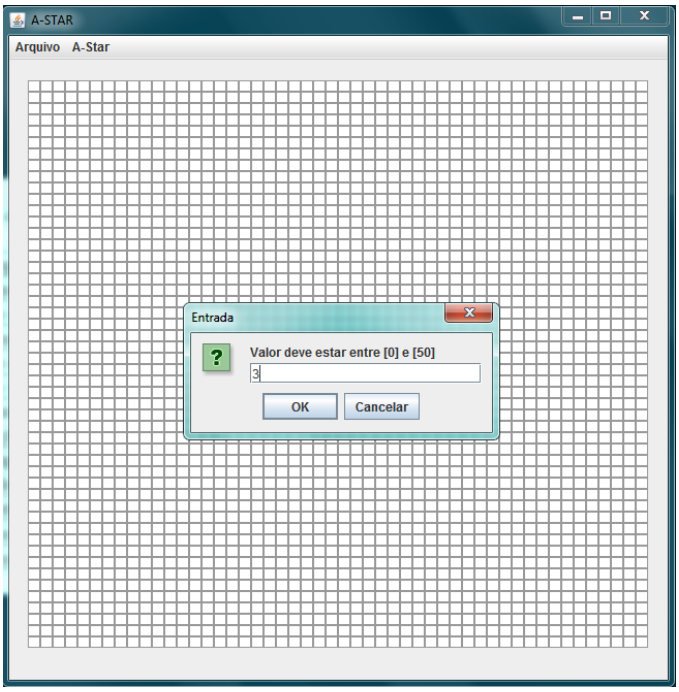

Figura 12 *– Definição da quantidade de obstáculos*

Em seguida, quando clicar no botão **OK**, esses obstáculos são representados pela cor preta. Os pontos de origem e destino são gerados automaticamente, representados respectivamente pela cor verde e azul. A Figura 13 mostra com clareza um ambiente quadrado 50 X 50, com três obstáculos em cada coluna.

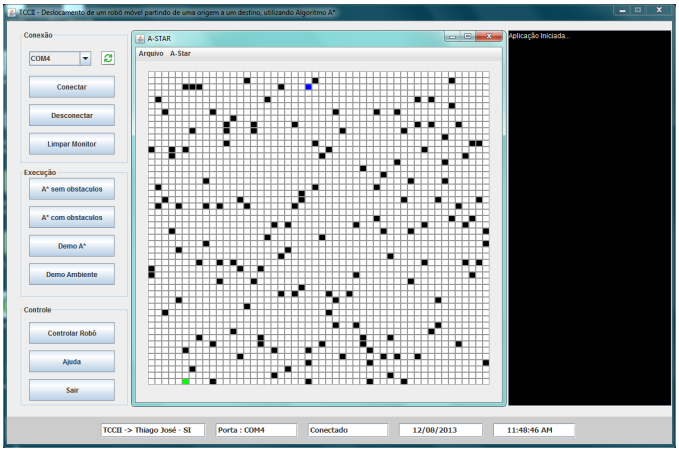

Figura 13 *– Matriz com obstáculos por coluna*

Com o ambiente definido, contendo os obstáculos, ponto de origem, e destino, o algoritmo $A*$ pode ser executado normalmente a partir da opção Menor Caminho. O menor caminho

é gerado a partir de um ponto origem a um ponto destino, com o auxílio de uma heurística. A Figura 14 abaixo mostra todos os pontos correntes que o algoritmo seleciona até chegar ao destino, pontos esses representados pela cor amarela, e o caminho representado pela cor vermelha. Nessa execução, o custo foi de 66, ou seja, para se deslocar do ponto origem até o destino, teve que percorrer 66 células, em um tempo de 38846555 nanossegundos e 38 milissegundos de execução. Esse tempo é calculado com o auxílio da função **System.nanoTime()**, sendo iniciado a partir do momento que começa a busca pelo menor caminho e ao término da busca é calculada a diferença do tempo final com o tempo inicial **(t2 - t1)**. O tempo inicial e final é armazenado em uma variável do tipo **long**. O valor da diferença entre os dois tempos é representado em nanossegundos. Para converter o valor de nanossegundos para milissegundos foi utilizada a função **TimeUnit.MILLISECONDS.convert((t2 - t1), TimeUnit.NANOSECONDS)**. Esses valores são mostrados no monitor serial localizado na lateral direita da tela principal da aplicação.

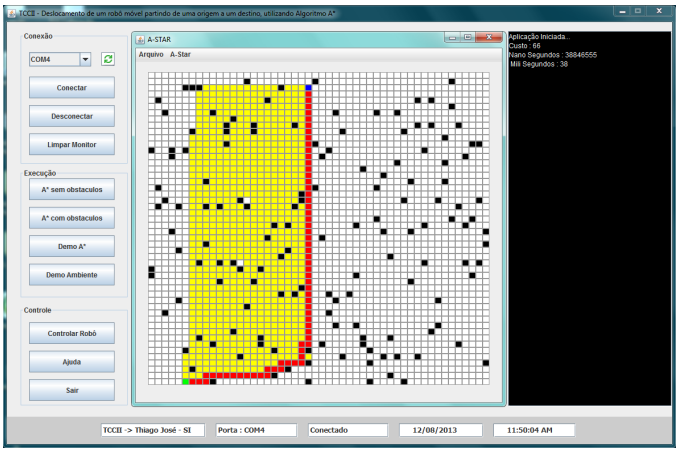

Figura 14 *– Menor caminho com três obstáculos por coluna*

O algoritmo *A** foi implementado para funcionar de forma dinâmica, para que possam ser definidos ambientes quadrados, onde os mesmos podem ser definidos no mínimo 3x3 e no máximo 80x80. Uma demonstração de um ambiente maior é apresentado na Figura 15, onde foi definida uma matriz 80x80 com obstáculos gerados aleatoriamente.

Para apresentação do robô funcionando, o ambiente real foi construído com material de isopor de tamanho quadrado de 75 cm, como uma matriz quadrada 3x3, contendo 9 células de 8 cm cada sendo que 3 são os obstáculos, e foi definido um ponto de origem e um ponto de destino. A origem está localizada na posição (0,0) e o destino na posição (2,2). O ambiente real foi projetado similar ao ambiente virtual (Figura 16), sendo definida uma matriz quadrada 3x3 com 3 obstáculos representado pela cor preta, ponto de origem na cor verde, o azul sendo o destino, e por fim o amarelo sendo o caminho que o robô deverá percorrer. A posição (2,0) não é um obstáculo, mas permanece sem acesso, ou seja, não tem como o robô chegar até essa

posição, pois não tem um caminho possível. Com base no ambiente virtual foi construído um ambiente real, com as mesmas características do virtual (Figura 17).

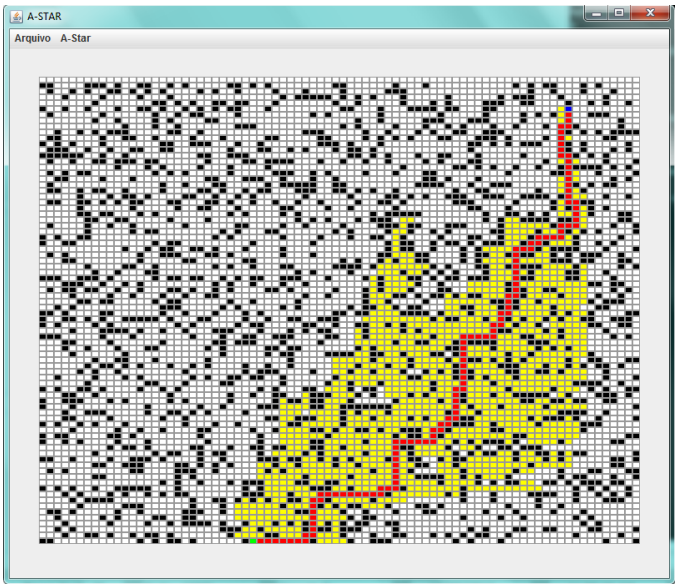

Figura 15 *– Menor caminho com obstáculos numa matriz 80x80*

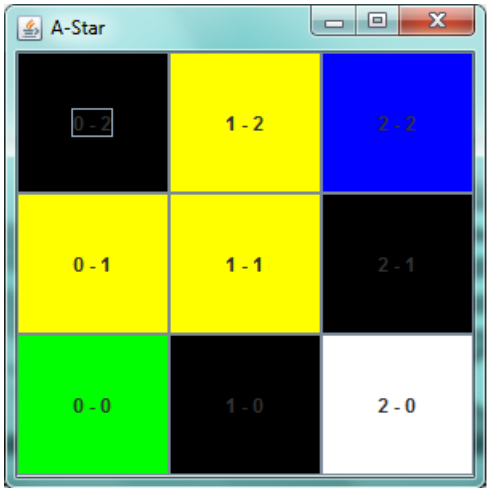

Figura 16 *– Ambiente virtual*

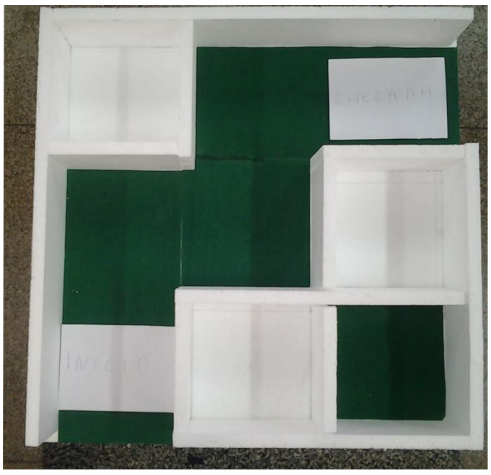

Figura 17 *– Ambiente real*

3.11 Controle

O projeto apresenta o deslocamento de um robô móvel de forma autônoma e como funcionalidade extra foi implementada uma funcionalidade de controle de robôs. Com a utilização dos módulos de RF, foi possível implementar o envio de informações em tempo real, informações essas que serão processadas pelo robô ao recebê-las. As formas principais de controle do robô é mover o mesmo para frente, para traz, direita, e esquerda, tendo como prioridade a execução dos dados recebidos pelo robô em tempo real e com esse controle de dados podem

utilizar mais funcionalidades no robô, tal como controlar um servo motor, executar músicas, dentre outras. A Figura 18 abaixo mostra a tela que é apresentada para controlar o robô após clicar no botão **Controlar Robô**, é uma interface bem intuitiva com recursos e funcionalidades.

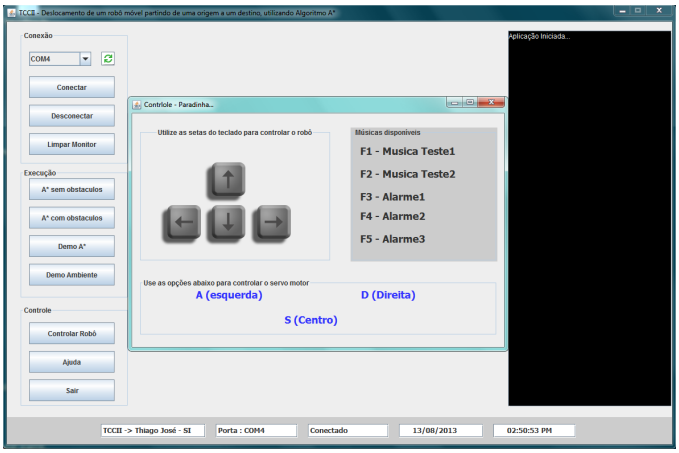

Figura 18 *– Controle do robô*

A implementação dessa interface foi baseado em um robô de 4 rodas, com a possibilidade de movimentação para quatro direções. O usuário pode utilizar as setas dos teclados para controle, utilizar as teclas F1 à F5 para executar as músicas e podendo também movimentar um servo motor usando as letras A, S, e D. Foi utilizada a opção **implements KeyListener** demonstrada na Tabela 3, pois a partir do evento (**KeyEventek**) a aplicação consegue capturar informações do teclado, como por exemplo, o **(ek.getKeyCode() == ek.VK_A)** que verifica se o valor obtido do teclado é igual a letra A.

EVENTO	VALOR ENVIADO	EXECUÇÃO
ek.getKeyCode() == ek.VK_SPACE	0	Centraliza rodas dianteiras
ek.getKeyCode() == ek.VK_A	1	Move servo para esquerda
ek.getKeyCode() == ek.VK_UP	2	Move robô para frente
ek.getKeyCode() == ek.VK_D	3	Move servo para direita
ek.getKeyCode() == ek.VK_LEFT	4	Move robô para esquerda
ek.getKeyCode() == ek.VK_RIGHT	6	Move robô para Direita
ek.getKeyCode() == ek.VK_S	7	Centraliza servo
ek.getKeyCode() == ek.VK_DOWN	8	Aciona a Ré
ek.getKeyCode() == ek.VK_F1	9	Música teste 1
ek.getKeyCode() == ek.VK_F2	10	Música teste 2
ek.getKeyCode() == ek.VK_F3	11	Alarme 1
ek.getKeyCode() == ek.VK_F4	12	Alarme 2
ek.getKeyCode() == ek.VK_F5	13	Alarme 3

Tabela 3 *– Eventos e execução do robô*

Ainda com base na opção de controle, temos os dois últimos botões. O botão **Ajuda** exibe uma tela com o nome do sistema, dados do desenvolvedor, orientadores e membros, como mostra na Figura 19. Ao final da opção de controle temos o botão **Sair** que tem como função encerrar a aplicação, mas antes de encerrar é feita uma pergunta para o usuário se realmente deseja encerrá-la.

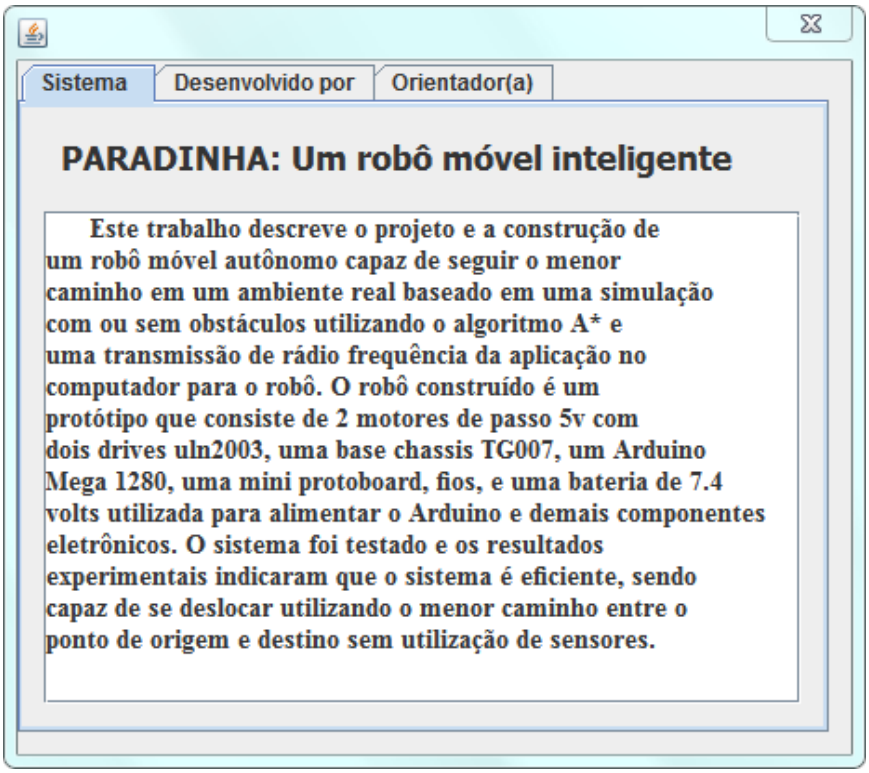

Figura 19 *– A Tela da opção Ajuda*

3.12 Transmissão

A transmissão das coordenadas do menor caminho acontece com o uso de dois Arduino e dois módulos de RF 433mhz. As coordenadas são transmitidas da aplicação para o Arduino, através da comunicação serial. Cada coordenada é transmitida individualmente, sendo que envia o valor de X e depois o de Y. Esses valores são adicionados sequencialmente em um vetor, quando os mesmos chegam ao Arduino que está conectado a aplicação.

A utilização da biblioteca **VirtualWire.h**, possibilita que dois módulos de RF possam estabelecer uma comunicação entre eles. O tipo de transmissão simplex, podendo ser transmitidas mensagens em apenas um sentido, do transmissor para o receptor (aplicação e robô). As coordenadas vindas da aplicação são adicionadas em um vetor para que possam ser transmitidas

para o robô de uma só vez, e no caso de falha é só transmitir novamente. Houve uma alteração na biblioteca **VirtualWire.h**, para que possa ser transmitido uma mensagem com maior quantidade de *bytes*, de no máximo 130 *bytes* por vez.

A quantidade de *bits* a ser transmitidos é definido a partir de **vw_setup(2000)**, nesse caso está configurado para transmitir dois mil *bits* a cada segundo. A função **vw_send((uint8_t *) vetor, tam)** é responsável pela transmissão da mensagem, usando dois parâmetros, o primeiro sendo a mensagem (**vetor**) e o segundo o tamanho da mensagem (**tam**), para enviar a mensagem deve utilizar o **vw_wait_tx()**. Isso acontece no lado do transmissor. O receptor usa a mesma lógica, o **vw_rx_start()** habilita o recebimento da mensagem, em seguida verifica se chegou alguma mensagem com o **vw_get_message(vetor, &tam)**, se retornar verdadeiro é porque chegou uma mensagem.

Como proposta no tema apresentado, a idéia de deslocar um robô de um ponto a outro desviando dos obstáculos percorrendo um menor caminho, não é bem simples de representar na prática, ou seja, em um ambiente real. Envolve uma série de fatores que contribuem para que o robô não faça o percurso pelo menor caminho corretamente. Mesmo com a execução eficiente do algoritmo de menor caminho, pode haver interferências na transmissão das coordenadas, perda de energia nas baterias que alimentam o robô ou até mesmo algum problema com os componentes eletrônicos. E isto pode interferir no bom desempenho do robô.

4 Testes realizados

Os testes foram realizados para verificar a eficiência da execução do algoritmo *A** com e sem obstáculos, e da interface gráfica responsável por exibir o ambiente gerado e o possível caminho de cada ambiente. Além disso, calcular a distância máxima em que os módulos de RF conseguem transmitir as coordenadas da aplicação para o robô. Para verificar a eficiência da execução do algoritmo *A** e da interface gráfica, foram realizados dois testes, ambos utilizando a função **System.nanoTime()** para obtenção do tempo de execução.

4.1 Eficiência do *A** sem obstáculos

O primeiro teste realizado verifica o tempo em que o *A** gasta para encontrar o menor caminho em ambientes sem obstáculos e o tempo gasto para exibi-lo na interface. O objetivo principal foi verificar se com a alteração do tamanho da matriz, o algoritmo consegue resolver o problema em tempo razoável. Além disso, averiguar o tempo máximo que as informações do menor caminho foram exibidas na interface gráfica. A Tabela 4 abaixo mostra os resultados obtidos com o teste, onde o mesmo foi realizado com seis matrizes quadradas de tamanhos diferentes, fixando o ponto de origem e destino de cada matriz como a primeira e última coordenada, respectivamente, para todos os ambientes testados. O cálculo de tempo da execução do *A** foi obtido em nanossegundos, e em seguida foi convertido em milissegundos, isso acontece também com o cálculo da interface. O tempo de execução do *A** com a interface é o cálculo da soma do tempo de execução do *A** com o tempo que leva para ser exibido o ambiente gráfico do caminho gerado. O custo de cada caminho gerado representa a quantidade de células entre a origem e o destino.

Matriz	A*		A* e interface		Custo
	ns	ms	ns	ms	
5x5	514007	0	1046637	1	9
10x10	1673780	1	2036006	2	19
15x15	4081325	4	4482660	4	29
20x20	6357574	6	6713747	4	39
25x25	14536053	14	14904331	14	49
30x30	31215197	31	31698010	31	59

***Tabela 4** – teste de eficiência em ambientes sem obstáculos.*

Observando os dados do teste de eficiência sem obstáculos (tabela 4), pode-se perceber que o tempo de execução cresce numa proporção menor que o tamanho da matriz gerada. Por

exemplo, enquanto o ambiente aumenta de 25 células (matriz 5x5) para 100 células (matriz 10x10) o tempo de execução em milissegundos aumenta de 1 para 2 milissegundos, pode-se ainda verificar que mesmo em ambientes maiores, o período de tempo de execução foi bem favorável, ou seja, em menos de um segundo o menor caminho foi encontrado e mostrado na interface gráfica em todos os ambientes testados. O mais importante é que a diferença de tempo entre a execução do A* e a exibição das coordenadas na interface gráfica é mínima, favorecendo assim bom desempenho de tempo de execução, da aplicação desenvolvida.

4.2 Eficiência do *A** com obstáculos

Os testes de eficiência do *A** com obstáculos foram realizados de forma semelhante ao teste descrito na seção anterior. O que diferenciou foi a presença de obstáculos neste teste. O cálculo de tempo de execução foi realizado duas vezes em cada matriz definida, prevalecendo os pontos origem e destino, mudando apenas as posições dos obstáculos. O propósito desse teste foi verificar as alterações do tempo de execução do *A** e da interface gráfica, além do custo com a mudança dos obstáculos no ambiente. O teste calcula o tempo de execução do *A** e o tempo que leva para que essas informações sejam exibidas na interface. A tabela 5 abaixo mostra os resultados obtidos com o teste, onde foram preservados os mesmos tamanhos de matrizes mostrados na Tabela 4, e acrescentados mais três matrizes quadrada: 40x40, 50x50 e 60x60.

Matriz	Obstaculos	A*		A* e interface		Custo
		ns	ms	ns	ms	
5x5	5	401791	0	935806	0	9
5x5	4	334748	0	703484	0	9
10x10	30	328231	0	692776	0	19
10x10	28	402722	0	794271	0	19
15x15	73	2250591	2	2631432	2	29
15x15	74	2246401	2	2668212	2	41
20x20	120	1206772	1	1516845	1	39
20x20	119	1097361	1	1781757	1	39
25x25	200	3514163	3	3890814	3	49
25x25	200	2020597	2	2812541	3	49
30x30	298	3880105	3	4258618	4	59
30x30	300	4652494	4	5013315	5	59
40x40	520	17691864	17	18089931	18	79
40x40	520	42733766	42	43155111	43	87
50x50	799	42256085	42	42575004	42	99
50x50	800	32846341	32	33255116	33	101
60x60	1200	70174311	70	70626385	70	119
60x60	1200	65177291	65	65536250	65	121

Tabela 5 *– Teste de eficiência em ambientes com obstáculos.*

Mesmo que o custo seja maior que outro em duas matrizes de mesmo tamanho, mudando apenas os obstáculos, não quer dizer que vai levar mais tempo para que o algoritmo encontre o menor caminho, por exemplo, a matriz 60x60 (Tabela 5), apresenta um custo de 121 e foi executado em menos tempo em relação ao custo 119, foi a mesma matriz com a mesma quantidade de obstáculos.

4.3 Comparação do A* com e sem obstáculos

Fazendo uma comparação entre os testes realizados em ambientes com e sem obstáculos, nota-se que em matrizes de tamanho quadrada de 5x5 até 30x30, o tempo de execução em ambientes com obstáculos foi menor do que ambientes sem obstáculos. Um exemplo claro é o da matriz 5x5, onde o tempo de execução apenas do A* em um ambiente sem obstáculos foi de 514007ns e a mesma matriz com obstáculos o tempo de execução foi de 401791ns, tendo uma diferença de 112216ns. A Tabela 6 mostra a continuação da análise de comparação entre ambientes com e sem obstáculos, listando o tempo que leva para o A* encontrar o possível caminho e exibi-los na tela.

Matriz	Com obstáculos		Sem obstáculos	
	ns	ms	ns	ms
5x5	935806	0	1046637	1
10x10	692776	0	2036006	2
15x15	2631432	2	4482660	4
20x20	1516845	1	6713747	4
25x25	3890814	3	14904331	14
30x30	4258618	4	31698010	31

***Tabela 6** – Comparação de eficiência do A* com e sem obstáculos*

4.4 Transmissão das coordenadas

O teste de transmissão foi realizado em um ambiente aberto da Universidade, onde foram utilizados dois Arduino, dois módulos de RF 433Mhz e um computador com processador Core 2 Duo, 4 Gb memória e sistema operacional Windows. O objetivo do teste foi verificar até que distância os módulos de RF consegue transmitir dados de um Arduino a outro. Nas especificações dos módulos diz que o alcance de transmissão é de 20 a 200 metros, só que para atingir a distância máxima vai depender do que possa gerar interferir na hora da transmissão. Na realização desse teste foram utilizados dois algoritmos, um para transmitir um vetor de coordenadas e outro pra recebê-las.

O Arduino transmissor ficava enviando um vetor de *bytes* a cada segundo, enquanto o receptor recebia e imprimia os dados recebidos no monitor serial do computador. O tamanho

máximo do vetor foi de 120 posições, a partir desse tamanho deve ser transmitido de duas vezes, neste caso podem ser transmitidas as coordenadas de no máximo 60 de custo. A cada cinco segundos os dois módulos foram se distanciando e atingindo uma distância de 7 metros sem perdas, sendo transmitido um vetor de tamanho 10 e o máximo 120. Essa distância de 7 metros em um ambiente aberto com pouca interferência, é a distância máxima que o robô poderá ficar do computador que está executando a aplicação para encontrar o menor caminho. No caso de falha no ato da transmissão das coordenadas para o robô, os dados podem ser retransmitidos novamente.

5 Considerações finais

Este trabalho propôs a implementação de um sistema robótico, utilizando simulação de ambientes quadrados, para roteamento de robô móvel capaz de seguir a trajetória do menor caminho encontrado entre os pontos de origem e destino definidos previamente em um ambiente com ou sem obstáculos. A metodologia utilizada foi experimental e separada em três fases: construção do robô (paradinha); a aplicação que simula o ambiente e calcula o menor caminho existente utilizando o algoritmo *A**; e, a transmissão entre a aplicação virtual e o robô no ambiente real usando Arduino.

O trabalho desenvolvido teve como resultado um robô móvel terrestre do tipo carro bastante eficiente. Para verificar a eficiência do sistema robótico, foram utilizados vários testes, desde a fase de construção do robô, utilizando simulações, comprovando o funcionamento do robô no ambiente real, fazendo a adequação da transmissão da aplicação para o robô, até a fase de testes de complexidade de tempo do algoritmo *A**.

Os resultados apresentados nos experimentos foram satisfatórios comprovando que o tempo de execução do algoritmo *A** crescia lentamente comparado com o aumento do tamanho da matriz de simulação do ambiente real. Com relação ao movimento do robô paradinha, verificou-se que existe uma pequena diferença em relação à simulação da aplicação, devido alguma a estrutura física do robô (componentes eletrônicos utilizados) e ao próprio ambiente montado e isto faz com que os movimentos não sejam exatamente 90^{o}. No entanto, a taxa de erro é tão pequena que é pouco perceptível.

Além disso, comparando o tempo de execução do algoritmo nos testes realizados com e sem obstáculos, pode-se perceber que o algoritmo encontra o menor caminho em menos tempo no ambiente com obstáculos. Isto acontece porque com os obstáculos o espaço de busca do algoritmo A* é reduzido, ou seja, são menos coordenadas para ser adicionados e removidos da lista corrente.

Referências

ARDUINO. *Arduino*. 2013. Disponível em: <http://www.arduino.cc>. Acesso em: 26 de Mar. 2013.

BOURG, David M.; SEEMANN, Glenn. *AI for Game Developers*. 2. ed. North Mankato: O'Reilly Media, 2004. 621 p.

CORMEN, Thomas H. et al. *ALGORITMOS Teoria e Prática*. 8. ed. Rio de Janeiro: Elsevier Editora Ltda, 2008.

DEITEL, P. J.; DEITEL, H. M. *Java: como programar*. 6. ed. São Paulo: Pearson Education do Brasil, 2006.

DEVMEDIA. *Utilizando a API RXTX para manipulação da serial*. 2013. Disponível em: <http://www.devmedia.com.br/utilizando-a-api-rxtx-para-manipulacao-da-serial-parte-i/6722>. Acesso em: 23 de Jun. 2013.

FERNANDES, A. M. R. *Inteligência Artificial: noções gerais*. 2. ed. Florianópolis: VisualBooks Editora, 2005.

GALDINO, C. H. S.; FARIA, R. P. *Uma aplicação de busca do menor caminho com obstáculos*. 2009. Artigo. Disponível em: <http://projetoia2009v2.googlecode.com/files/Apresentacao_IA.pdf>. Acesso em: 21 de Jan. 2013.

NETBEANS. *The NetBeans Platform*. 2013. Disponível em: <https://netbeans.org/features/platform/index.html>. Acesso em: 26 de Mar. 2013.

ORACLE. *ORACLE Java*. 2013. Disponível em: <http://www.oracle.com/us/technologies/java/overview/index.html>. Acesso em: 26 de Mar. 2013.

PATEL, Amit. *Introduction to A**. 2013. Disponível em: <http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html#the-a-star-algorithm>. Acesso em: 26 de Mar. 2013.

RUSSELL, S; NORVIG, P. *Inteligência Artificial*. 5. ed. Rio de Janeiro: Elsevier, 2004.

TANENBAUM, Andrew S.; WETHERALL, David. *Redes de Computadores*. 5. ed. São Paulo: PRENTICE HALL, 2011.

WOLF, D. F. et al. *Robótica móvel inteligente: Da simulação às aplicações no mundo real*. 2009. Disponível em: <http://osorio.wait4.org/publications/2009/CL_JAI2009_Completo.pdf>. Acesso em: 25 de Ago. 2013.

ZIVIANI, N. *Projeto de Algoritmos com implementação em Java e C++*. 2. ed. São Paulo: Thomson Learning Ediçøes Ltda, 2007. 621 p.

ANEXO A – Algoritmo de Relaxamento, Bellman-Ford e Dijkstra

Algoritmo de relaxamento

```
RELAX(u, v, w)
1  if d[v] >d[u] + w(u, v)
2    then d[v]← d[u] + w(u, v)
3 π[v] ←u
```

Algoritmo de Bellman-Ford

```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2 for i ←1 to |V[G]| -1
3       do for cada aresta (u, v) ϵ E[G]
4               do RELAX(u, v, w)
5 for cada aresta (u, v) ϵ E[G]
6       do if d[v] >d[u] + w(u, v)
7               then return FALSE
8  returnTRUE
```

Algoritmo de Dijkstra

```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S ← 0
3  Q ← V[G]
4  while Q ≠ 0
5       do u← EXTRACT-MIN(Q)
6               S ←S U {u}
7               for cada vértice v ϵ Adj[u]
8                       do RELAX(G, w, s)
```

ANEXO B – Algoritmo A*

1. Adicione o nó inicial à lista OPEN.
2. Enquanto a lista OPEN não estiver vazia, faça:
    a) O nó corrente será o nó de menor custo da lista OPEN.
    b) Se o nó corrente for igual ao nó destino, então o caminho estácompleto e o algoritmo termina.
    c) Senão, mova o nó corrente da lista OPEN para a lista CLOSED e examine cada nó adjacente ao nó corrente.
    d) Para cada nó adjacente, faça:
        Se o nó não estána lista OPEN e não estána lista CLOSED e não é um obstáculo, então:
            Mova-o para a lista OPEN e calcule o valor de f(n).

Onde OPEN representa a lista dos nós que ainda não foram analisados e CLOSED a lista dos nós já visitados.