XXVII JORNADAS DE JOVENS PESQUISADORES
23 A 25 DE OUTUBRO DE 2019
A ciência e a tecnologia na produção de inovação e transformação social

22. Tecnologias da Informação e da Comunicação

Orientação de um robô autônomo para limpeza de espelhos d'água genéricos de forma otimizada

Autor: Costa, Bruno; bruno.costa105@hotmail.com
Orientador: Batista, Valério; valerio.batista@ufabc.edu.br
Centro de Matemática, Computação e Cognição
Universidade Federal do ABC

Resumo

Empregados em tarefas diversas, os robôs tendem a ficar cada vez mais inteligentes e autônomos. Entretanto, algumas tarefas que são triviais para os seres humanos podem ser desafiadoras para um robô, pois ensiná-lo a processar sinais visuais ou a pensar como nós é uma tarefa complexa. Dentro desse contexto, o projeto pode ser visto como uma etapa do desenvolvimento de um robô limpador de espelhos d'água otimizado, cujo objetivo é elaborar, implantar e testar metodologias para a sua orientação, fazendo com que ele consiga processar a rota otimizada e percorrê-la conforme for ordenado.

O protótipo utilizado é um robô LEGO NXT e a primeira etapa do projeto consistiu na escolha da linguagem de programação ao lado de um estudo de como utilizá-la no robô. Após a obtenção de um mapa de contorno do espelho d'água desejado, um projeto anterior do professor orientador o quadricula, gerando vértices e logo após a rota otimizada. O robô, por sua vez, deve processar esses dados e percorrer a rota, recebendo como entrada as coordenadas dos vértices e a ordem a ser percorrida. O algoritmo desenvolvido consiste em ler esses dados, transformá-los em vetores e então fazer com que o robô consiga percorrê-los independentemente da posição ou orientação em que ele se encontrar utilizando uma lógica que envolve o quadrante de cada vetor obtido dos dados. A última etapa foi o desenvolvimento de um modelo para obtenção da energia gasta após todo o trajeto, analisando e extraindo dados da tensão de saída da bateria enquanto o robô repete um mesmo movimento várias vezes. Os principais resultados obtidos foram o algoritmo ser validado, pois com vários testes o robô de fato conseguiu percorrer uma dada rota desejada, e a obtenção de um valor de energia gasta coerente, apesar de pouco precisa.

Palavras chaves: programação, protótipo, robótica

Introdução

Empregados nas mais diversas tarefas, os

robôs tendem a ficar cada vez mais inteligentes e autônomos. Entretanto, algumas tarefas completamente triviais para os seres humanos, como por exemplo apanhar uma bola arremessada, podem ser desafiadoras para um robô realizar. Isso porque ensinar um robô a processar sinais visuais ou pensar como nós é uma tarefa complexa, então temos que a automatização e otimização de um processo requer um grande estudo. Neste projeto, temos como objetivo o estudo de metodologias para a orientação de um robô, ou seja, como fazê-lo percorrer uma dada rota, sem qualquer desvio e de forma completamente autônoma. O robô terá como entrada de dados um grafo com uma rota para a limpeza de um espelho d'água, sendo esta rota a mais otimizada possível. O primeiro passo para a orientação do robô é a obtenção de uma malha baseada no mapa do espelho d'água, que pode ser obtido por exemplo através de uma planta baixa. A malha conterá todas as possíveis trajetórias que o robô pode realizar, sendo fundamental uma boa malha para maior otimização da limpeza. Em um projeto anterior [1], foi desenvolvido com sucesso um algoritmo que dado um mapa de contornos, é construída uma malha adequada (para detalhes, ver [1]). Após a obtenção da malha, é necessário definir qual rota o robô percorrerá, sendo que esta rota definida por um grafo foi obtida a partir de um projeto de pesquisa anterior [3]. Assim, será necessário definir métodos para o robô percorrer este grafo otimizado em um espelho d'água sintético, métodos estes trabalhados e estudados nesse projeto.

Objetivos

A robótica é uma ciência intrínseca ao nosso dia a dia, e os seus avanços proporcionam cada vez mais facilidades e uma maior qualidade de vida para as pessoas. O nosso projeto pode ser visto como uma etapa do desenvolvimento de um robô limpador de espelhos d'água otimizado, em que objetivamos elaborar, implantar e testar metodologias para a sua orientação, fazendo com que ele consiga processar a rota otimizada e percorrê-la conforme for ordenado e realizar um estudo do seu consumo de bateria a partir de um modelo a ser desenvolvido. Dessa forma, este projeto tem como objetivos:

- Gerar a rota que o robô deve fazer a partir do seu mapa de contorno, utilizando o algoritmo genético de um projeto anterior [2];
- Desenvolver o algoritmo para o robô ler e percorrer a rota, programando-o em Matlab e realizando testes;
- Desenvolver um modelo para analisar o gasto de energia;
- Comparar o consumo real de bateria com os resultados teóricos apresentados em [2];

Materiais e Métodos

- Materiais

Os materiais a serem utilizados são:

- Kit LEGO Mindstorm;
- Computadores para desenvolvimento do código;

Sendo o kit disponibilizado pelo orientador e os computadores pela UFABC, totalizando um custo zero.

- Escolha da linguagem de programação

A primeira etapa do projeto foi a definição da linguagem de programação a ser utilizada. O robô lego NXT possui uma linguagem gráfica nativa, o NXT-G, entretanto ela não atende às necessidades desse projeto, como por exemplo a leitura de dados de arquivos de diferentes extensões ou um processamento matemático adequado, sendo necessária a utilização de uma linguagem mais robusta e flexível. Com isso, foi realizado um estudo acerca da viabilidade de se utilizar 3 linguagens de programação: C, Python e Matlab. Essas foram as opções por conta do fator limitante de ser necessário existir alguma biblioteca ou interface que realize a interação dessas linguagens com o robô, pois como ele possui um software padrão, o uso delas pode ser visto como uma adaptação e não algo nativo. Na linguagem C houve a opção de utilizarmos o NXC (Not Exactly C) [4], que foi desenvolvido especialmente para a programação dos robôs LEGO NXT. Também era possível a utilização do NXT-Python v3 [5], um pacote desenvolvido para possibilitar o controle do NXT através do Python. Por fim, poderia ser utilizada uma toolbox desenvolvida por uma universidade alemã, a RWTH-Mindstorm NXT Toolbox, que permite o controle do robô através do Matlab. O Python foi descartado pelo fato do pacote ter informações muito confusas a respeito de seu funcionamento, sendo opções melhores o NXC e Matlab. Entre as duas opções, o Matlab foi escolhido pelo fato do site da toolbox ter uma documentação muito bem detalhada sobre o seu funcionamento, por existir a publicação do professor orientador fundamentada nessa toolbox e, por fim, pelo fato do Matlab ser uma linguagem confortável para o desenvolvimento do algoritmo do robô, que envolve uma série de funções matemáticas.

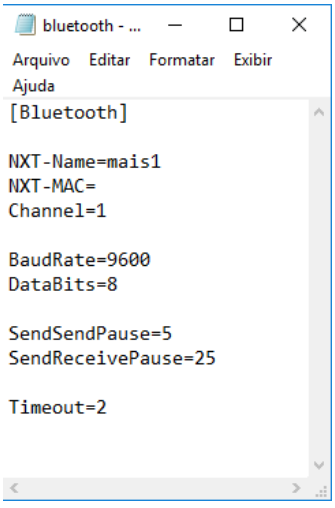
- Toolbox e a comunicação Bluetooth

Uma toolbox no Matlab é um pacote de arquivos que incluem códigos, apps, dados, exemplos e uma documentação, sendo extremamente útil para a realização de diversas tarefas. A RWTH - Mindstorm NXT Toolbox nos permite controlar o robô LEGO NXT com uma série de comandos pré estabelecidos, sendo isso que realiza a comunicação do Matlab com o robô. O processamento dos dados ocorre inteiramente no computador e só depois é enviado com funções já definidas para o robô, como por exemplo acionar um motor ou receber os dados de algum sensor. Tendo em vista que o conceito de

funcionamento da toolbox e do computador ser o controle remoto do robô, é necessário de alguma forma manter uma comunicação contínua entre os dois. Para isso existem duas opções: fazer a comunicação por USB ou por Bluetooth. Utilizar a comunicação USB seria a opção mais simples, porém o robô ficaria limitado ao comprimento do cabo, o que torna essa opção inviável. Para comunicação via Bluetooth seria necessário um notebook com Bluetooth que suporte o SPP (Serial Port Profile), requisitos atendidos e que portanto não foram um problema. Com isso, para iniciar a comunicação Bluetooth foi necessário primeiramente parear o notebook com o NXT. Após isso, utiliza-se o comando "*COM_MakeBTConfigFile*" para criar um arquivo de extensão *ini* com os parâmetros de configuração necessários para realizar a comunicação, conforme a imagem abaixo:

Figura 1: Parâmetros de configuração da comunicação.

Com isso, basta utilizar os comando



Com_OpenNXT e *COM_SetDefaultNXT* para realizar a comunicação Bluetooth entre o notebook e o robô, e então prosseguir com a programação.

- Acionamento dos motores

O primeiro passo para a orientação do robô é o acionamento dos motores. Em um vídeo tutorial, disponibilizado em [6], é ensinada uma forma de mover o robô para frente ou para trás, acionando dois motores simultaneamente. Isso requer a criação de um objeto motor, isto é, uma instância da classe *NXTMotor*. Esses objetos são capazes de enviar ou receber dados de um ou mais motores, sendo no projeto utilizados apenas para enviar dados como a velocidade, ângulo que o motor precisa rotacionar e a porta a qual está conectado. A base da orientação do robô consiste nas rotinas de movimentação e rotação. O grafo consiste de várias linhas retilíneas com diferentes angulações, então o robô necessita de uma rotina para andar uma certa distância retilínea, e outra para que ele possa rotacionar e então se movimentar em qualquer direção desejada.

- Rotina de movimentação

A primeira função desenvolvida no projeto foi a rotina de movimentação, que tem como entrada a distância que o robô deve percorrer (em centímetros) e a velocidade, que pode ser positiva ou negativa, sendo que a positiva faz o robô mover-se para

frente e a negativa faz com que ele se mova para trás.

De forma a controlar a distância que o robô percorre, é enviado para ele o ângulo de rotação do motor, por exemplo, enviar um ângulo de 360 graus fará com que sua roda dê uma volta completa, fazendo com que a distância percorrida seja igual a do perímetro de sua roda. Pensando nisso, para que ele se mova uma certa distância *x*, é possível fazer uma regra de três:

$$\frac{2 * \pi * raio}{x} = \frac{360}{ang} \rightarrow ang = \frac{360 \times x}{2 \times \pi \times raio}$$

Utilizando essa equação é possível converter uma distância *x* no ângulo (ang) que deve ser enviado para o motor, e ele rotacione o suficiente para que o robô percorra essa distância. O raio da equação é igual ao raio da roda do robô, equivalente a 2,8cm. A imagem abaixo mostra o código desenvolvido para essa rotina, que é utilizada muitas vezes no programa principal:

Editor - C:\Users\HP\Desktop\RWTHMindstormsNXT\IC\MyProgramFunctions\WalkNXT.m*

vectortraj.m | WalkNXT.m* | RotateNXT.m | readvert.m | readtraj.m | main.m

```
function WalkNXT(distance,speed)
%This function makes the robot walk in the distance and speed
%wanted.
%The inputs are the distance, that must be in centimeters, and the speed,
%that varies from -100 to 100.
%If the speed is positive, the robot will walk forward, if it is negative,
%it'll drive backward.
%Example: WalkNXT(17.59,100)

radius = 2.8;
perimeter = 2*pi*radius;
angle = int64((360/perimeter)*distance);

motorAB = NXTMotor();
motorAB.Port = 'AB';
motorAB.SpeedRegulation = 0;
motorAB.Power = -speed;
motorAB.TachoLimit = angle;

motorAB.SendToNXT();
motorAB.WaitFor();

end
```

Figura 2: Rotina para movimentação do robô.

- Rotina de rotação

A segunda função desenvolvida foi a de rotação, com uma lógica um pouco mais complexa. Os seus dados de entrada são o ângulo de rotação e a velocidade desejada. A rotação do robô ocorre sem que ele se desloque, pois não existe uma maneira de mudar sua orientação enquanto ele se desloca. O robô NXT é do tipo *2WD* (Two-wheel drive), ou seja, possui tração em duas de suas três rodas. A toolbox permite o envio de comandos distintos para os motores de cada roda de forma simultânea, o que implica na possibilidade de fazer um motor rotacionar uma roda na mesma velocidade que o outro, porém em sentidos opostos. Dessa forma, o robô não se desloca e apenas muda sua orientação, sendo isso a base da rotina de rotação.

Na imagem abaixo vemos o raio da roda do NXT que trataremos como R_1 e ao lado o raio do círculo cujo diâmetro é a distância entre os centro das rodas, que trataremos como R_2.

Figura 3: Raio da roda e raio entre as rodas.

Devido à rotina de movimentação, temos que para andar uma distância *x*, precisamos enviar para o motor um ângulo θ tal que

$$\theta = \frac{360 \times x}{2 \times \pi \times R_1} \qquad (1)$$

Tratando agora do círculo entre as rodas, temos que para mudar a orientação do robô em um ângulo α precisamos andar um arco de distância *x*, e como as rodas irão rotacionar uma no sentido oposto da outra o robô não irá sair de seu centro. Com isso a distância que for solicitada para ele percorrer será percorrida em um arco de um círculo com raio R_2 Utilizando a fórmula do arco, temos então que para o robô rotacionar um ângulo α, precisa percorrer um arco de distância *x* tal que

$$x = \frac{2 \times \pi \times R_2 \times \alpha}{360} \qquad (2)$$

Substituindo (2) em (1) concluímos que

$$\theta = \frac{R_2}{R_1} \times \alpha$$

Com isso, tem-se que para rotacionar um ângulo α, o ângulo que deve ser enviado para os motores rotacionarem deve ser igual à razão do raio das rodas e do raio da roda multiplicado pelo ângulo desejado. A imagem abaixo mostra o código desenvolvido para essa rotina, que é utilizada diversas vezes dentro do código principal, visto que tem a função de mudar a orientação do robô, tendo o detalhe de não poder ser executada simultaneamente com a de movimentação mostrada no tópico anterior.

Editor - C:\Users\HP\Desktop\RWTHMindstormsNXT\IC\MyProgramFunctions\RotateNXT.m*

vectortraj.m | WalkNXT.m | RotateNXT.m* | readvert.m | readtraj.m | main.m

```
function RotateNXT(angle,speed)
%This function makes the robot rotate the degrees given in the input in the
%wanted speed.

r = (5.5/2.8);
x = angle*r;

motorA = NXTMotor();
motorA.Port = 'A';
motorA.Power = (-1*speed);
motorA.TachoLimit = (int64(x));

motorB = NXTMotor();
motorB.Port = 'B';
motorB.Power = speed;
motorB.TachoLimit = (int64(x));

motorA.SendToNXT();
motorB.SendToNXT();

motorA.WaitFor();
motorB.WaitFor();

end
```

Figura 4: Rotina para rotação do robô.

- Programa principal

Para programar orientação do robô é necessário primeiramente o processamento do grafo e da rota. O grafo contém todos os vértices, enquanto que a rota contém a ordem dos vértices que o robô deve percorrer. O projeto conta com uma rotina para a leitura do arquivo contendo o grafo e armazenamento das coordenadas de seus vértices em uma variável struct. Como entrada ela deve receber uma variável do tipo char com o nome do arquivo, incluindo a sua extensão.

Editor - C:\Users\HP\Desktop\RWTHMindstormsNXT\IC\MyProgramFunctions\readvert.m*

vectortraj.m | WalkNXT.m | RotateNXT.m | readvert.m* | readtraj.m | main.m

```
function vtx = readvert(filename)

warning('off','all')
arq=filename;
fid=fopen(arq,'rt');
lis=textread(arq,'%s');
fclose(fid);
delete arq;
i=1;
k=5;
while ~strcmp(lis(k),'edges')
 vtx(i).id = str2double(lis(k));
 vtx(i).y  = str2double(lis( k+2));
 vtx(i).z  = str2double(lis(k+3));

 if strcmp(lis(k+4),'original')
     k=k+6;
 else
     k=k+4;
 end
 i=i+1;
end
delete lis;
end
```

Figura 5: Rotina para leitura e

armazenamento dos dados do grafo.

A leitura da rota é feita por uma rotina semelhante, cuja entrada deve ser do tipo char e conter o nome do arquivo contendo a trajetória. A diferença é que a última linha do arquivo deve conter o caractere "\", e os dados são armazenados em um vetor.

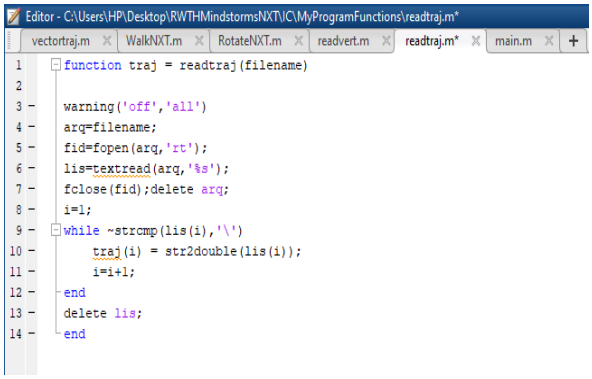


Figura 6: Rotina para leitura e armazenamento da rota.

Após a obtenção das coordenadas dos vértices e a trajetória que o robô deve executar, é possível criar os vetores que serão utilizados em sua orientação. Para a geração destes, desenvolveu-se uma rotina que tem como entrada uma variável struct contendo as coordenadas dos vértices e um vetor contendo a trajetória, ou seja, a ordem dos vértices a serem percorridos. Essas entradas podem ser obtidas através das duas rotinas explicadas nos tópicos anteriores, sendo essa separação de rotinas feita para uma melhor organização do código. O código dessa rotina é demasiadamente extenso para ser incluso no relatório, mas resumidamente ele calcula a diferença das coordenadas *x* e *y* dos vértices *n* e *n+1* (sendo *n* dado pelo vetor contendo a trajetória) para a obtenção dos catetos cuja hipotenusa é o vetor a ser percorrido. O seu módulo é igual ao valor da hipotenusa, sendo obtido através da equação de Pitágoras, e seu ângulo com relação ao eixo *x* é obtido através da tangente da razão entre os catetos oposto e adjacente. Por fim, utilizando-se as coordenadas é armazenado o quadrante a que pertence cada vetor, informação importantíssima para a lógica de sua orientação.

A lógica para a orientação dos vetores requer todos os passos anteriores e é pautada no quadrante dos vetores. Essa parte do código faz *n* iterações, tal que *n* é igual ao número de vetores criados. Cada iteração irá fazer com que o robô primeiro rotacione o ângulo desejado com a rotina de rotação e então percorra o módulo do vetor com a rotina de movimentação. Como citado anteriormente, o problema se encontra no fato de robô mudar sua direção a cada iteração, e o único ângulo que possuímos dos vetores é com relação ao eixo *x*. Para contornar esse problema, foi utilizado o quadrante do vetor anterior à movimentação para localizar qual a atual orientação do robô e o quadrante do vetor do próximo deslocamento para saber qual o ângulo que ele deveria rotacionar. Escrevendo caso a caso, como por exemplo estando com a orientação no primeiro quadrante e tendo

que rotacionar para o terceiro, foi possível determinar quantos graus o robô deveria rotacionar em função dos ângulos com relação ao eixo *x* do vetor anterior e do vetor subsequente, sendo totalizados 27 casos. Desses 27 casos, foi possível simplificar o código e gerar apenas 8 casos. A imagem abaixo mostra um caso, em que o robô está no primeiro quadrante e precisa se mover um ângulo ainda no primeiro quadrante maior do que o seu atual, sendo que o valor a ser enviado para a rotina de rotação é a subtração dos ângulos com o eixo x, $\alpha - \beta$. O valor a ser enviado para a rotina de movimentação deve ser equivalente ao módulo do vetor que faz um ângulo α com o eixo x, que por sua vez já foi calculado na criação dos vetores.

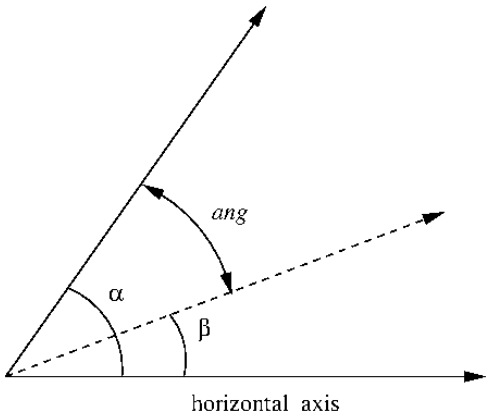


Figura 7: Exemplo de um caso do programa principal.

- Modelo para obtenção da energia gasta

Após obter sucesso na orientação do robô, a segunda parte da iniciação é o desenvolvimento de um modelo para analisar o consumo de energia do robô, e então comparar com gastos previstos teoricamente. A primeira etapa foi a pesquisa de como funcionam as baterias em geral, e um dado amplamente utilizado para monitorar o consumo é a tensão de saída emitida. A toolbox possui uma função chamada "*NXT_GetBatteryLevel()"* que retorna o valor, em mV, da tensão de saída no momento em que a função foi executada. A relação que a tensão de saída possui com o consumo de bateria é que ele serve como um indicador do quanto de energia uma bateria ainda possui, tendo valores mais elevados para uma certa faixa de tensão de saída e valores menores para outras faixas. Apesar de ser algo bem comum de se encontrar, como por exemplo em qualquer celular que mostra o quanto de energia ainda resta, a relação entre ambas não é linear e varia completamente de bateria para bateria, não existindo uma lei geral para monitorar a energia restante de uma bateria. De acordo com baterias comerciais do mercado, é possível em geral encontrar dados dessa relação (em alguns casos até mesmo gráficos) em um datasheet. No caso do robô, LEGO NXT, temos uma bateria que foi fabricada em 2006 e que não é produzida hoje em dia. Contatando a LEGO foi possível receber via email um datasheet do modelo de bateria que se encontra no robô, entretanto não havia nele uma relação da tensão de saída com a energia restante. Uma informação útil que pôde ser extraída do datasheet é a tensão de saída nominal e

a capacidade em mAh da bateria, o que permite calcular a energia total da bateria em joules. Tendo-se em vista que não haveria nenhuma forma de prever a relação entre a tensão de saída e a energia restante, foi desenvolvido um modelo. A hipótese inicial desse modelo é a de que o robô, ao realizar um mesmo movimento, iria sempre consumir o mesmo valor de energia da bateria, sendo uma hipótese plausível pois em um caso ideal é o que realmente acontece. Com essa hipótese em mente, foi desenvolvido um programa que fizesse com que o robô girasse 360º e então armazenasse sua tensão de saída. Após ter a bateria totalmente carregada, foi executado esse programa até a bateria se esgotar. Fazendo-se esse procedimento 3 vezes e realizando a média entre os valores obtidos após um mesmo movimento, foi-se obtido 483 valores de tensões de saída, desde a bateria com carga cheia até a bateria sem carga. O modelo presume que quando o robô não consegue ser ligado pela bateria, a sua energia restante é de zero joules. Como a hipótese inicial é a de que o robô sempre consome a mesma quantidade de energia em um mesmo movimento, o gasto de energia será sempre o mesmo após cada rotação, de forma que a porcentagem de energia decresça de forma linear, sendo possível dessa forma atribuir um valor de porcentagem de energia restante a cada valor de tensão de saída, e assim conseguindo uma relação entre ambas. Abaixo há o gráfico obtido com a relação da porcentagem de bateria e a tensão de saída dos 483 valores obtidos experimentalmente.

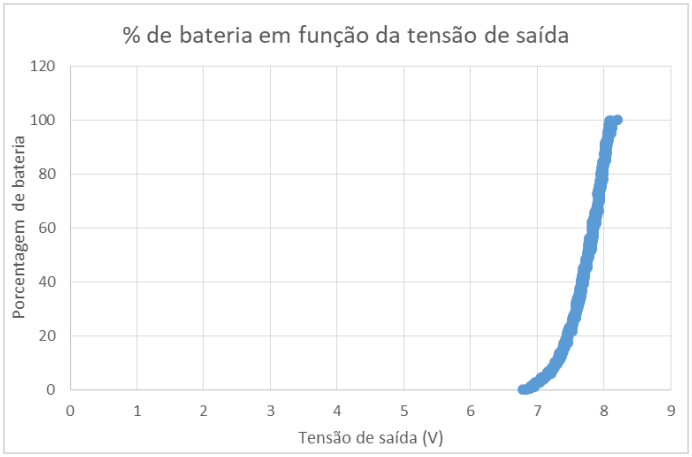


Figura 8: Gráfico da relação entre a porcentagem de energia restante da bateria e a sua tensão de saída.

Após a obtenção dessa relação, o próximo passo é encontrar uma função que receba de entrada o valor da tensão de saída, e devolva como saída a porcentagem de energia restante da bateria, com base nesses dados. Entretanto antes de qualquer tentativa foi notório o fato de que havia um tremendo ruído na informação bruta dos 483 valores, apesar da visualização do gráfico parecer uma função bem uniforme, sendo que esse ruído acumularia um erro muito grande na hora de fazer uma interpolação. Por conta disso foi pesquisado alguns métodos para realizar uma filtragem dos dados, sendo o método escolhido o filtro de média móvel e aplicado no software Excel. Apesar de não ser um dos filtros mais

refinados, após executá-lo duas vezes com intervalos de dois valores, e uma vez com intervalos de quatro valores, o ruído foi suficientemente amenizado.

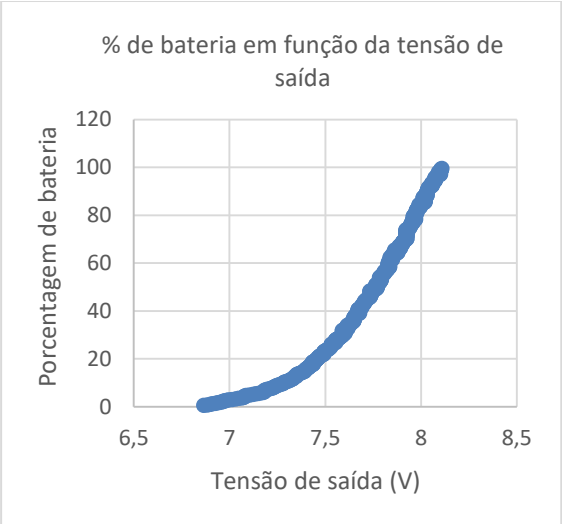


Figura 9: Gráfico da relação entre a porcentagem de energia da bateria e a tensão de saída filtrado.

Para encontrar uma função, primeiramente foi utilizado o método de interpolação polinomial de Lagrange, programado em Matlab. O grande problema encontrado nesse método é que com poucos pontos (até 10), ele fornece valores coerentes porém pouco precisos, e com muitos pontos (mais que 10) ele fica muito preciso para alguns pontos entretanto com um erro da ordem de 10^{31} para outros dentro do intervalo. Isso ocorre porque o método polinomial de Lagrange, apesar de forçar o polinômio interpolador de passar nos pontos dados, tem o problema de gerar um polinômio de *n-1* grau, sendo *n* o número de pontos fornecidos. Com isso em vista, um polinômio de 20º grau por exemplo pode gerar um valor muito fora do esperado por possuir grandes valores de expoentes e não estar necessariamente perto de alguns pontos que estão no alcance da tensão de saída da bateria do robô. A segunda tentativa foi a implementação do método dos mínimos quadrados, em que se encontra uma função que não necessariamente passa pelos pontos dados, mas está o mais próxima possível deles. Dessa forma foi possível incluir todos os pontos dos dados filtrados, e com um polinômio de 10º grau foi obtido uma função que relacionasse a tensão de saída com a porcentagem de bateria de uma forma a não ter um erro tão grande. Há situações em que se encontra 105% ou -3%, por exemplo, entretanto esse erro é intrínseco do ruído dos dados obtidos e do próprio método dos mínimos quadrados, sendo que o filtro de média móvel e o polinômio com um grau alto são formas de buscar minimizar esses erros.

- Rotina para obtenção da energia restante

Tendo a relação entre porcentagem de energia da bateria e a sua tensão de saída, é possível descobrir a energia restante em joules tendo-se como entrada a tensão de saída da bateria. Primeiramente é necessário descobrir quantos joules tem na bateria quando ela está com 100% de energia restante. Esse valor pode ser obtido através da multiplicação da tensão nominal de 7.4 V pela capacidade de 1,4 Ah:

$$Energia = 7.4\frac{J}{C} \times 1{,}4\frac{C}{s} \times 3600s = 37296\,J$$

Com isso, através de uma regra de 3 simples pode-se obter a energia restante de energia para uma dada porcentagem, sendo que 100% equivale a 37296 joules, então y% equivale a x joules. A rotina para obtenção da energia se chama "*remainingenergy*", sendo que nessa rotina está incluso o método dos mínimos quadrados com todos os dados obtidos através do filtro de média móvel, e com a porcentagem obtida e a regra de três obtém-se a quantidade de energia restante em joules para uma dada entrada de tensão em mV. Combinando esse função junto da "*NXT_GetBatteryLevel()*", que retorna a tensão de saída da bateria em mV, é possível obter a quantidade de energia restante da bateria em qualquer momento. Para a obtenção da energia gasta em um movimento, foi incluído no programa principal a energia restante na bateria antes e após um movimento, e então subtraída uma da outra. Assim pode-se obter tanto a energia gasta movimento a movimento como a energia gasta em uma limpeza total.

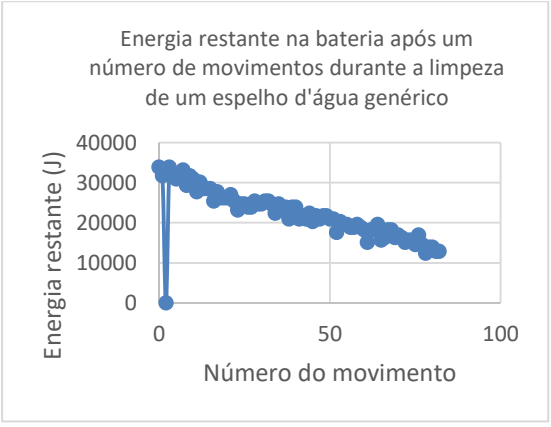
Resultados e Discussão

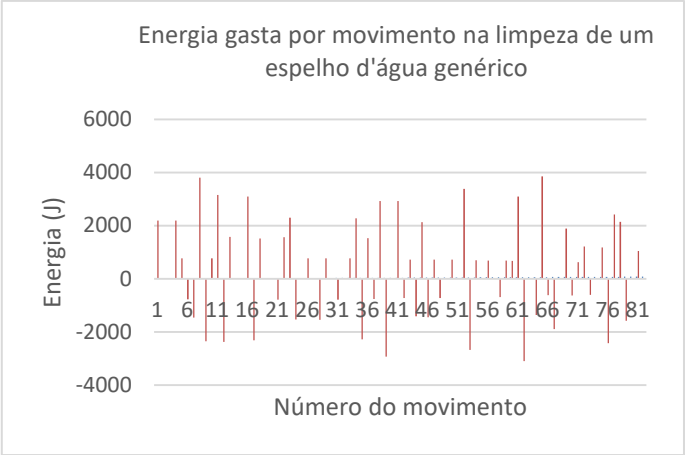
Os primeiros testes realizados com o programa principal foram feitos utilizando-se um grafo menor do que o espelho d'água da UFABC. Após algumas tentativas e correções, o programa funcionou corretamente com este grafo, com o robô percorrendo a rota desejada de acordo com os seus vértices. Em contrapartida descobrimos um problema, a rotina de rotação que é perfeita na teoria, tem uma margem de erro na prática. Como os grafos possuem vários vértices, o robô acaba por executar muitas rotações, acumulando um erro de poucos graus a cada movimento que causa no fim um grande desvio da rota esperada, sendo uma falha mecânica e não da programação. Em um projeto de robótica, é possível diminuir esse erro criando uma rotina de rotação utilizando os sensores giroscópio e acelerômetro, pois eles fariam uma leitura real de quantos graus o robô girou e o erro associado seria ao erro do sensor, que por sua vez seria muito menor. Foi feita uma extensa pesquisa para ver se seria possível encontrar esses sensores para o robô, entretanto pelo fato do LEGO NXT ser um robô antigo não foi possível encontrar uma venda desses sensores por fontes confiáveis, sendo descartada essa possibilidade. Após a confecção do grafo do espelho d'água da UFABC o programa também funcionou perfeitamente nele, entretanto como ele é um grafo muito extenso com mais de 1000 coordenadas, o erro acumulado se tornou muito grande, sendo optado por continuar trabalhando e realizando os testes no grafo menor utilizado no começo da iniciação. Entretanto, o programa funciona para qualquer espelho d'água, basta ter as coordenadas e a rota. Com os testes foi constatado que o robô

sempre parava a execução de seu movimento ao percorrer um dado vetor, mesmo que não precisasse realizar alguma rotação. Com isso, foi feita uma alteração no programa principal para ele identificar quando não for necessário rotacionar e prosseguir seu caminho retilíneo somando os módulos dos vetores com mesmo ângulo, para os percorrer de uma só vez.

Outro resultado interessante, mas já apresentado na metodologia, foi o gráfico obtido da relação entre a tensão de saída da bateria e a porcentagem de energia restante, pois partindo de uma hipótese o modelo resultou em um gráfico muito uniforme e com formato parecido com o de outras baterias comerciais, o que realça a ideia de que o comportamento real é próximo do modelado.

Quanto a testes, utilizando o grafo menor de um espelho d'água sintético citado anteriormente, foi obtido o valor da energia gasta movimento a movimento e a variação da energia restante, conforme é mostrado nos gráficos abaixo.





Analisando esses gráficos, pode-se observar que o gasto de energia durante a limpeza obteve valores coesos, visto que apesar do enorme ruído na obtenção da tensão de saída, houve um decrescimento contínuo da quantidade de energia restante. Quanto a energia gasta movimento a movimento, pode-se observar claramente o ruído na existência de alguns valores negativos, o que seria um absurdo pois é como se o robô ganhasse energia após executar um movimento. Entretanto, logo pode-se perceber que a energia gasta após um movimento retilíneo está em geral entre 1000 e 4000 joules, variando dependendo da distância percorrida, e as rotações gastam cerca de 700 joules. Esses valores são muito diferentes do que os supostos em [2], na prática a energia gasta está na ordem de 10^3 enquanto o teórico está na ordem de 10^1, o que não é problemático visto que em [2] esses valores supostos servem apenas para mensurar uma diferença de valor teórico gasto entre diferentes rotas, e que não tinham valores empíricos como os apresentados neste projeto.

Conclusões

Foi obtido resultados muito positivos, com o robô percorrendo as rotas corretamente e obtendo-se valores coesos de energia gasta. Como se trata de um protótipo, o desvio que o robô tem da rota original não é preocupante, pois isso seria facilmente resolvido com a robótica do projeto real. A hipótese para o desenvolvimento do modelo de consumo de energia da bateria se mostrou bem próxima de um comportamento real, visto que foi obtido uma curva bem similar a outras baterias comerciais que possuem essa relação bem definida. Apesar de o trabalho ter sido feito com um espelho d'água sintético, o código gerado funciona para qualquer tipo de espelho d'água, sendo feito da forma mais genérica possível. O método para a obtenção da energia gasta é específico para essa bateria, mas seguindo a metodologia descrita é possível desenvolve-lo para qualquer outro tipo de bateria.

Referências Bibliográficas

[1] Batista, V.R., Zampirolli, F.A., & Gonzalez, J.A.Q. (2016). An Optimal Cleaning Robot For All Kinds of Reflection Pools. Santo André, SP: Universidade Federal do ABC.

[2] Batista, V.R., & Zampirolli, F.A. (2018). Optimizing Robotic Pool-Cleaning with a Genetic Algorithm. *Journal of Intelligent & Robotic Systems*.

[3] Batista, V.R., & Zampirolli, F.A. (2018). Optimising Robotic Pool-Cleaning with a Genetic Algorithm. *Journal of Intelligent & Robotic Systems*, 1-16.

[4] Brickxcc. (2006). Welcome to Next Byte Codes, Not eXactly C, and SuperPro C. Recuperado de http://bricxcc.sourceforge.net/nbc/.

[5] GitHub. (2015). NXT-Python. Recuperado de https://github.com/Eelviny/nxt-python.

[6] MathWorks. (2014, 26 de março). MATLAB Meets MINDSTORMS: How to Control LEGO NXT Robots Using MATLAB for Educational Purposes. Recuperado de https://www.mathworks.com/videos/matlab-meets-mindstorms-how-to-control-lego-nxt-robots-using-matlab-for-educational-purposes-90418.html.