CARLOS EDUARDO DE ALMEIDA

NAVEGAÇÃO DE ROBÔ MÓVEL UTILIZANDO DECOMPOSIÇÃO DO AMBIENTE EM CÉLULAS E FUNÇÕES POTENCIAIS

Trabalho de Conclusão de Curso apresentado
à Escola de Engenharia de São Carlos, da
Universidade de São Paulo

Curso de Engenharia Elétrica com ênfase em
Eletrônica

ORIENTADOR: Valdir Grassi Junior

São Carlos

2011

Ficha catalográfica preparada pela Seção de Tratamento
da Informação do Serviço de Biblioteca – EESC/USP

Almeida, Carlos Eduardo de

A447n Navegação de robô móvel utilizando decomposição do ambiente em células e funções potenciais / Carlos Eduardo de Almeida ; orientador Valdir Grassi Júnior -- São Carlos, 2011.

Monografia (Graduação em Engenharia Elétrica com ênfase em Eletrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2011.

Em memória de minha mãe, Maria Teresinha Cabelo, que sempre me proporcionou carinho e amor.

Agradecimentos

Gostaria de agradecer aos meus pais, pois sem eles nunca teria a chance de fazer uma universidade pública como a USP. A minha esposa, que sempre me encorajou a alcançar tudo que tenho em minha vida hoje. E ao Professor Valdir, que me ajudou muito nesse trabalho de conclusão de curso.

‘With great Power comes great responsibility’
Bejamin “Ben” Parker

Resumo

O objetivo deste trabalho de conclusão de curso foi implementar um método de planejamento de movimento baseado em decomposição do ambiente em células e funções de navegação, conforme proposto em Conner et al (2003). Para implementar o método proposto, foi utilizado o software Matlab® para desenvolver um conjunto de ferramentas que calcula a função de navegação de um ambiente dividido em células. Com isso, este método permitirá que o robô navegue pelo ambiente a partir de qualquer posição inicial até uma posição final desejada desviando dos obstáculos, que foram considerados no mapa no momento do planejamento.

Palavras-Chave: Planejamento de movimento. Funções de Navegação. Robô Móvel.

Abstract

The objective of this work was to implement a method of motion planning based on decomposition of the environment in cells and navigation functions, as proposed in Conner et al (2003). Matlab was used to implement the proposed method, to develop a set of tools that calculates the navigation function of an environment divided into cells. Therefore, this method will allow the robot to navigate the environment from any initial position to a desired final position avoiding the obstacles considered on the map at the motion planning step.

Keywords: Planning of movement. Navigation functions. Mobile robot.

Lista de Figuras

Sumário

1. Introdução

Em tarefas de navegação autônoma, o planejamento de movimento tem o propósito de determinar quais movimentos o robô deve realizar de forma que alcance posições ou configurações desejadas no ambiente sem que ocorram colisões com obstáculos (Latombe,1991). No processo de planejamento utilizam-se informações sobre o ambiente no qual o robô está inserido, na forma de um mapa, juntamente com informações sobre o próprio robô, ou seja, seu modelo cinemático e dinâmico. Diversos métodos de planejamento para navegação são apresentados de forma detalhada em Latombe (1991), Choset et al (2005), e LaValle (2006), dentre outros. Algumas das abordagens utilizadas para planejamento de movimento são: *roadmaps*; decomposição em células; e campos potenciais. A escolha do método de planejamento influencia a maneira como ocorre a navegação e também como comportamentos reativos de navegação podem ser integrados no sistema de controle. Um exemplo de comportamento reativo é o desvio de obstáculos móveis que não são considerados no mapa no momento do planejamento.

Em Conner (2007) utiliza-se o método de decomposição do ambiente em células combinado com o método de função potencial de navegação. Esta técnica divide o espaço livre do ambiente em células de geometria simples conectadas umas as outras através de uma fronteira. A partir desta divisão, uma vez que a posição de destino final para o robô é especificada, determina-se possíveis seqüências de células pelas quais o robô deve caminhar para que chegue à célula que contêm a posição final. Então, uma função potencial de navegação é definida dentro de cada célula do ambiente. Isto é feito de forma que cada função dentro de uma célula conduz o robô, pelo gradiente negativo da função, até a fronteira dessa célula com a próxima célula da seqüência. Uma vez que o robô atravessa a fronteira da célula em que está, ele cai no domínio de convergência de outra função potencial de navegação. Isso é feito sucessivamente, até que a função de navegação na última célula conduza o robô até a posição de destino desejada.

Outra forma de navegar usando esse método é definir uma rota de monitoramento para o robô através de uma seqüência específica de células. Assim, o robô realiza o percurso continuamente sendo levado de uma célula a outra, conforme ilustrado na Figura 1. A vantagem em utilizar o método de planejamento de movimento baseado em divisão do

ambiente em células e funções potenciais é a de que ao dividir o ambiente em células, o cálculo de uma função potencial é feito em uma região menor e de geometria mais simples que a do ambiente todo. Também como a navegação se baseia em funções potenciais, podem-se aproveitar as diversas técnicas conhecidas na literatura para desvio de obstáculos que se baseiam nessas funções.

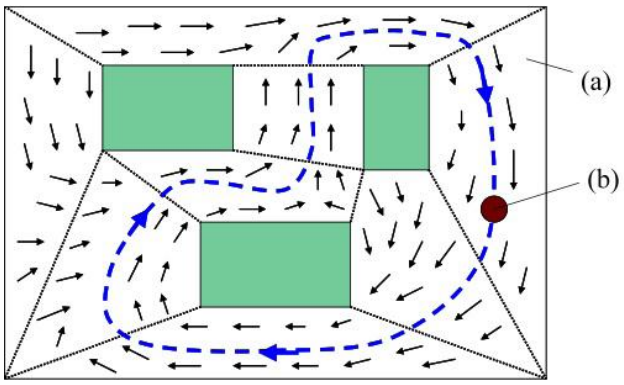

Figura 1: Ilustração de navegação do robô através do método de decomposição em células.

1.1. Objetivo

Este trabalho consiste no estudo e implementação do método proposto em Conner (2007) e implementá-lo utilizando o software Matlab®, este método tem como resultado uma matriz construída a partir dos valores da função de navegação para cada ponto do ambiente discretizado. A matriz em questão, por sua vez, pode ser utilizado por softwares de controle para movimentar o robô dentro do ambiente em questão.

1.2. Organização

O presente trabalho esta organizado da seguinte forma:

- O Capítulo 2 apresenta de forma breve alguns métodos de planejamento de movimento;
- O Capítulo 3 explica de forma detalhada o método utilizado neste trabalho de conclusão de curso;

- O Capítulo 4 contém as informações sobre a implementação do método de seqüência de funções de navegação e os resultados alcançados;
- O Capítulo 5 apresenta as conclusões relativas ao trabalho realizado neste projeto.

2. Planejamento de Movimento

O método implementado por esse trabalho de conclusão de curso tem como objetivo resolver problemas de navegação de robôs móveis, sendo assim, este capítulo apresenta de forma sucinta diversos métodos de planejamento de movimento para fins de navegação, tais como: *roadmaps*, decomposição em células e campos potenciais; os quais são descritos mais detalhadamente em Latombe (1991), Choset et al. (2005), e LaValle (2006). Para se explicar os métodos mencionados acima, primeiro é necessária uma explicação sobre o espaço de configurações, que será dada na próxima seção.

2.1. Espaço de configurações

O espaço de configurações é uma maneira de representar o meio ambiente no qual o robô está inserido. Porém, para se entender essa definição de espaço de configurações é necessária algumas explicações sobre os seguintes conceitos: espaço de trabalho, representação de um robô no espaço, obstáculos no ambiente, e por fim, configuração do ambiente.

O espaço de trabalho de um robô é o espaço por onde o robô se movimenta (meio físico) e pode ser definido no $\mathbf{R^3}$ ou no $\mathbf{R^2}$. No caso desse trabalho, é utilizado o $\mathbf{R^2}$ como espaço de trabalho, pois o robô utilizado irá se deslocar apenas no plano Euclidiano. No entanto, para robôs que se deslocam em três dimensões, normalmente é utilizado o espaço Euclidiano tridimensional para se definir o espaço de trabalho, aqui representado por $\mathcal{W}$.

Um subconjunto compacto do espaço de trabalho $\mathcal{W}$ representa um robô, $\mathcal{R}$, naquele espaço físico. Para um subconjunto ser compacto, este conjunto deve ser fechado e limitado, ou seja, deve possuir todos os pontos incluindo o limites/fronteiras do subconjunto. Um exemplo de um subconjunto compacto seria no **R,** [2,5]; e para um subconjunto não compacto, seria algo do tipo (2,5].

Obstáculos são subconjuntos fechados no espaço de trabalho $\mathcal{W}$ e serão representados por $\mathcal{B}_i,...,\mathcal{B}_q$. Definem-se, como os sistemas de coordenadas cartesianas, $\mathcal{F}w$ e $\mathcal{F}r$ em $\mathcal{W}$ e $\mathcal{R}$ respectivamente.

Finalmente a configuração do ambiente, representada por q, é definida como sendo a completa especificação da posição de todos os pontos do robô relativa ao sistema fixo de coordenadas do meio ambiente, $\mathcal{F}w$. Para certa configuração do ambiente, q, o subconjunto de espaço de trabalho, $\mathcal{W}$, ocupado pelo robô, $\mathcal{R}$, é representado por $\mathcal{R}(q)$.

Com as definições introduzidas acima, é possível definir o espaço de configurações para um robô. O espaço de configurações, C, de um robô é definido como sendo o conjunto de todas as configurações possíveis para este robô. No espaço de configurações o robô é definido como um ponto q. O espaço de configurações é uma ferramenta que vem sendo muito utilizada para formulação de problemas de planejamento de movimento e, um dos motivos, é que toda geometria da tarefa pode ser mapeada no espaço de configurações.

Podem-se definir facilmente dois subconjuntos distintos pertencentes ao espaço de configurações, sendo eles muito úteis para a formulação de problemas de planejamento de movimento: o espaço de configurações ocupado por obstáculos, $C_{\mathbf{obs}}$, e espaço de configurações livres, $C_{\mathbf{livre}}$.

O espaço de configurações ocupado por obstáculos é definido como sendo o conjunto de configurações para as quais há intersecção entre o robô e os obstáculos. O espaço de configurações livres, por sua vez, é definido como sendo o espaço de configurações que não há intersecção entre o robô e os obstáculos.

É possível determinar $C_{\mathbf{obs}}$ e $C_{\mathbf{livre}}$ a partir de um mapa do ambiente de trabalho que represente os obstáculos e da geometria do robô, por exemplo, para um robô circular que realiza apenas movimentos de translação em um plano. A configuração do robô é dada pelas coordenadas do ponto de referência localizado no seu centro, $q = \{x, y\}$. O espaço de configurações para este robô é bidimensional e pode ser representado no plano Euclidiano, $\mathbf{R}^2$. Além disso, para este robô o espaço de trabalho também é o plano Euclidiano. Entretanto, é importante ressaltar que, mesmo tendo a mesma dimensão, o espaço de configurações, C, é diferente do espaço de trabalho, $\mathcal{W}$. Para definir o espaço de configurações ocupado pelos obstáculos, move-se o robô ao redor de cada obstáculo do espaço de trabalho conforme mostra a Figura 2. A trajetória descrita pela referência do robô, $\{x, y\}$, define a fronteira do obstáculo em C. Dessa maneira, determina-se a restrição

que o obstáculo estabelece sobre a configuração do robô. Para este caso, os obstáculos no espaço de configurações equivalem aos obstáculos no ambiente de trabalho expandidos pela dimensão radial do robô circular. Na Figura 2, C_{obs} é representado pela áreas em cinza e preto. O espaço de configurações livres, C_{livre}, corresponde ao complemento de C_{obs} em C.

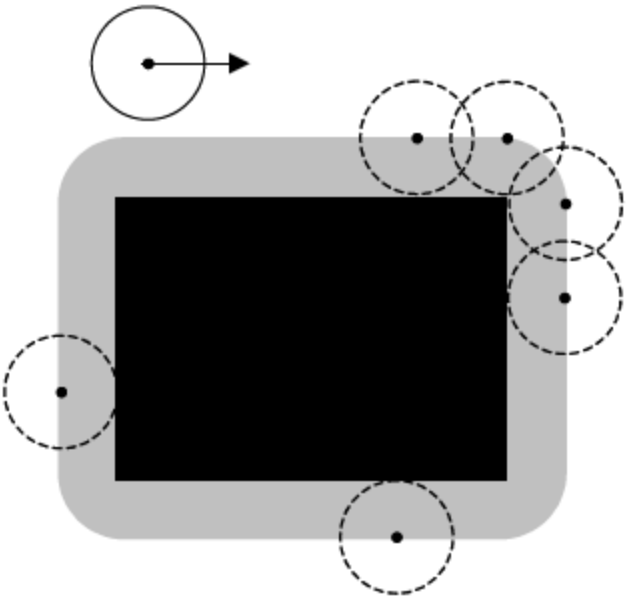

Figura 2: Espaço de configurações de um obstáculo quadrado para um robô circular. A área preta representa o obstáculo no ambiente de trabalho e a área cinza, o obstáculo no espaço de configurações.

Feita essa introdução sobre espaço de configuração, é possível entender os métodos de planejamento de movimento, os quais serão descritos abaixo.

2.2. Planejamento utilizando *roadmaps*

Uma *roadmap* como o próprio nome sugere é um "mapa de ruas", sendo essas ruas criadas a partir de determinadas regras (métodos). Dentre os métodos para se criar uma *roadmap* aqui será citado o método de grafo de visibilidade e o método de retração (diagrama de Voronoi generalizado).

O método de grafo de visibilidade apenas é utilizado para planejamento de movimento em espaços de configurações bidimensionais com obstáculos (C_{obs}) poligonais. Neste método impõe duas regras para se criar a *roadmap*:

- Um nó do grafo é ligado a outro se estes dois nós estão no campo de visão um do outro;
- Cada ponto do espaço de configurações livre (C_{livre}) deve estar no campo de visão de pelo menos um nó do grafo de visibilidade.

Com ambas as regras mencionadas acima é possível acessar a *roadmap* de qualquer lugar do espaço de configurações. Por fim o grafo de visibilidade é criado ligando-se dois vértices dos obstáculos (nós) por uma linha reta que não passe pelo interior de nenhum obstáculo. A Figura 3 mostra um exemplo de um grafo de visibilidade, sendo os polígonos rachurados, os obstáculos.

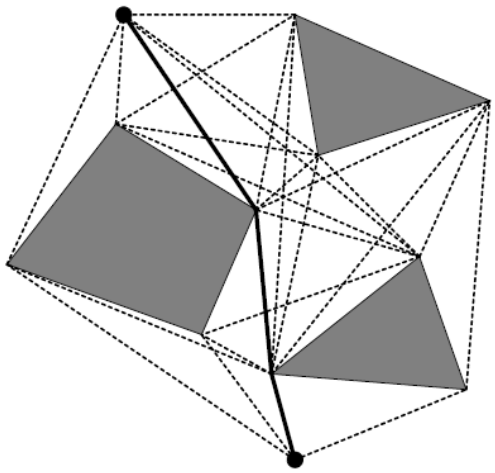

Figura 3: Exemplo de um grafo de visibilidade.

O método de retração consiste em construir uma *roadmap* a partir da retração de seu espaço de configurações livres no diagrama de Voronoi generalizado, sendo este diagrama um subconjunto de C_{livre} que maximiza a distância entre robô e os obstáculos. Na Figura 4 é mostrado um exemplo do método de retração.

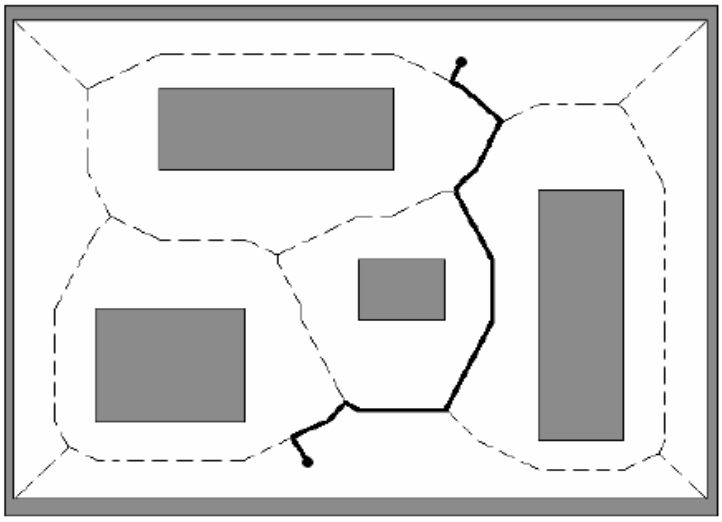

Figura 4: Exemplo de um diagrama generalizado de Voronoi (método de retração).

O planejamento de movimento que usa as *roadmaps* respeita duas regras. A primeira é encontrar um caminho que conecte a posição inicial, $q_{inicial}$, e a posição final, q_{final}, a *roadmap*. Seja $q'_{inicial}$ e q'_{final}, respectivamente, o ponto de conexão de $q_{inicial}$ e q_{final} a *roadmap*. Então, a segunda regra é procurar um caminho na *roadmap* que ligue os dois pontos de conexão $q'_{inicial}$ e q'_{final}.

2.3. Decomposição em células

Decompor o espaço de configurações livres em células (subconjuntos de C_{livre}) pode ajudar na tarefa de encontrar uma solução no planejamento de movimento do robô, pois dentro de cada célula a trajetória entre dois pontos é facilmente gerada.

Quando a intersecção de duas células não é nula, isso significa que estas duas células são adjacentes, pois ambas possuem uma mesma fronteira. Esta relação de adjacência entre as células pode ser representada por um grafo não direcional. Com o espaço de configurações livres dividido em células e o grafo de adjacência construído, é possível obter o planejamento de movimento seguindo três etapas:

- Determinam-se quais células possuem a posição inicial e final do robô;
- Através do grafo de adjacência faz-se uma busca para determinar um caminho de

células intermediarias entre as células que possuem a configuração inicial e final do robô;

- E por fim, determina-se dentro de cada célula da solução encontrada, uma curva que ligue dois pontos entre as fronteiras dessas células.

A Figura 5 mostra um exemplo de decomposição em células, sendo o tracejado representando a fronteira das células.

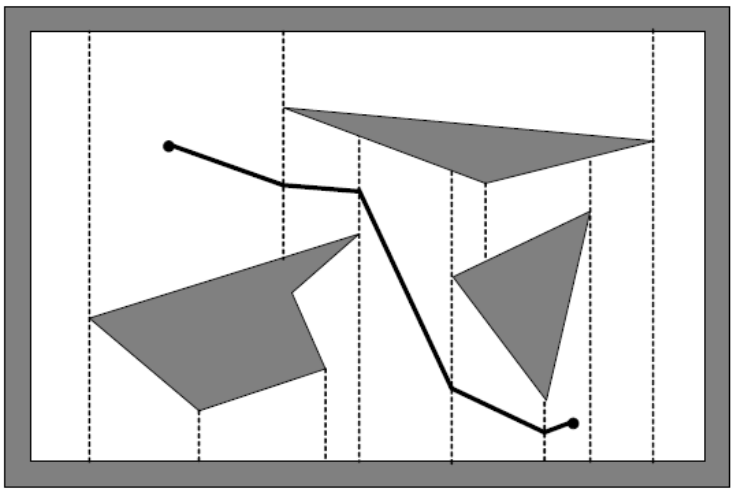

Figura 5: Exemplo de decomposição em células.

2.4. Campo Potencial

Nos métodos de campo potencial, também chamada de função potencial, o robô move-se de acordo com o gradiente negativo de um campo potencial definido no ambiente (espaço de configurações). Neste caso, o robô é considerado como um ponto no espaço de configurações livres.

Uma abordagem de campos potenciais é atribuir um potencial atrativo para a posição final do robô e um potencial repulsivo aos obstáculos. O potencial atrativo é independente dos obstáculos e o potencial repulsivo é independente da posição final, sendo, a soma desses dois campos potenciais de atração e de repulsão, o campo potencial que age sobre o robô.

O problema desse método é que não há garantia que o robô alcance o seu objetivo, pois na maioria dos métodos baseados em campos potenciais, o robô pode ficar preso em mínimos locais, os quais são pontos (diferentes do ponto de objetivo do robô) onde o resultado da soma dos potenciais de atração e de repulsão é igual zero. No entanto, existem algumas técnicas para lidar com esses mínimos locais, fazendo o robô sair deles, uma dessas técnicas é a função de navegação.

2.4.1. Função de navegação

Rimon e Koditschek (1992) desenvolveram um tipo especial de função potencial que é livre de mínimos locais, também chamada de função de navegação. Para se calcular essa função de navegação, é necessário conhecimento prévio de todo o ambiente em que o robô está inserido, em outras palavras, ter o conhecimento do espaço de configurações livres e dos obstáculos do ambiente de trabalho.

Para tanto, define-se uma função de navegação da seguinte forma (Rimon & Koditschek, 1992):

Seja C_{livre} o espaço de configurações livres do robô, e q_{obj} a configuração final desejada para o robô dentro de C_{livre}. Um mapeamento $\phi : C_{\text{livre}} \rightarrow [0,1]$ é uma função de navegação se ela é:

- Suave em C_{livre}, ou seja, ela possui derivadas de segunda ordem contínuas;
- Polar em q_{obj}, ou seja, possui um único mínimo em q_{obj} no componente conectado de C_{livre} que contém q_{obj};
- Admissível em C_{livre}, ou seja, uniformemente máxima na fronteira de C_{livre};
- E seja uma função do tipo Morse.

Uma função do tipo Morse é uma função cujos pontos críticos (pontos em que a derivada ou o gradiente da função é zero) não são degenerativos. Isso significa que os pontos críticos são isolados, e que quando o robô segue o gradiente negativo de uma função desse tipo, qualquer perturbação é capaz fazer com que o robô saia do mínimo local.

Um exemplo de uma função de navegação calculada para um ambiente composto de esferas é mostrada na Figura 6, onde o ponto branco é o q_{obj}.

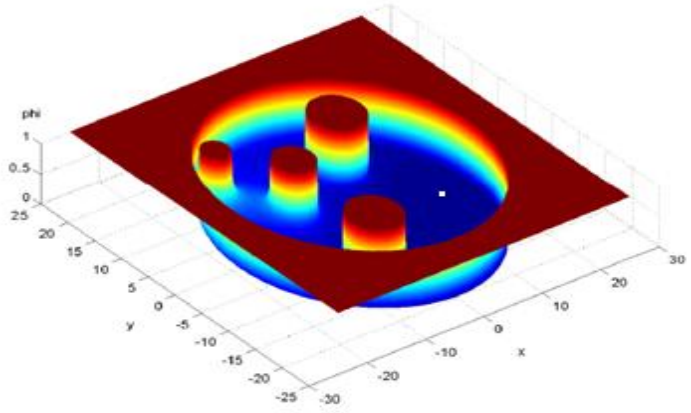

Figura 6: Função de navegação em um ambiente composto de esferas.

Um ambiente composto apenas de esferas não representa muito a realidade de grande parte dos problemas de planejamento de movimento, sendo assim, Rimon e Koditschek (1992) demonstram que se um ambiente composto de esferas pode ser mapeado de modo difeomórfico[1] em um ambiente de geometria mais complexa, existe uma função de navegação definida neste ambiente composto de esperas que também pode ser mapeada para o ambiente com geometria mais complexa preservando suas propriedades de função de navegação.

Como o método utilizado para esse trabalho de conclusão de curso utiliza essa técnica, será feita uma explicação mais detalhada em comparação com os métodos explicados anteriormente, por isso o próximo capitulo é dedicado a explicar o método utilizado nesse trabalho de conclusão de curso.

[1] Uma função difeomórfica é uma função suave, bijetora, e que possui uma inversa também suave.

3. Seqüência de funções de navegação

Neste trabalho, foi implementado o método de navegação baseado em seqüência de funções de navegação proposto por Conner (2007). Esse método consiste em combinar o método de decomposição em células com o método de função de navegação.

A primeira etapa desse método é dividir o ambiente de trabalho em regiões menores (células) e, nesse caso, no formato de polígonos convexos. Logo após, define-se a posição final do robô, para determinar a seqüência de células que o robô deve passar para atingir o seu objetivo. Até esse ponto não há diferença entre esse método e o método de decomposição de células explicado no capítulo anterior. No entanto, o método começa a se diferenciar nesse ponto, onde, para cada célula, é definida uma função de navegação. Essas funções de navegação são muito mais simples de se calcular, pois não são funções de navegação para ambientes inteiros. Esse fato é verdadeiro por causa da geometria simples dos polígonos convexos.

As funções de navegação de cada célula, exceto da célula que possui a posição final do robô, são definidas de tal forma que leve o robô até a fronteira de uma célula adjacente, e quando o robô chega à fronteira, ele imediatamente entra no domínio da função de navegação dessa célula adjacente. Assim, o robô vai navegando de célula a célula até chegar à célula que possui a sua posição final. Para esta célula, a função de navegação definida é diferente das demais, sendo definida para convergir para a posição final do robô, onde é o único mínimo em todo o ambiente de trabalho.

Em linhas gerais, para calcular as funções de navegação mencionadas anteriormente, primeiro, cada célula é mapeada difeomórficamente para um disco no plano Euclidiano. Depois, aplica-se uma transformação para variáveis polares. A partir daí calcula-se a função potencial no disco polar, e como foi aplicado um mapeamento difeomórfico para “transformar” o polígono convexo no disco, essa função de navegação pode ser mapeada para o polígono convexo preservando suas propriedades. Cada uma dessas etapas será detalhada a seguir.

3.1. Mapeando polígonos convexos em discos

Para se mapear um polígono em um disco, primeiro é necessário saber como se pode definir um polígono no espaço $\mathbf{R}^2$, sendo assim, esta seção será dedicada a explicar matematicamente como definir um polígono no plano Euclidiano e, em seguida, mapeá-lo em um disco.

No plano Euclidiano, $\mathbf{R}^2$, são necessárias no mínimo três restrições espaciais (segmentos de reta, lados do polígono) para se definir um polígono. Cada lado do polígono pode ser representando por um ponto médio, p, e uma normal, n, com o sentido da normal apontando para fora do polígono. Para um ponto no plano Euclidiano, q, define-se a distancia do ponto q ao *i-ésimo* segmento de reta como:

$$\beta_i(q) = -n_i \cdot (q - p_i) \qquad \text{(Eq. 3.1)}$$

Uma condição necessária para o conjunto $\{(p_i,n_i) \mid \boldsymbol{i} = 1,...,m\}$ de segmentos de reta definir um polígono válido é:

$$\forall \boldsymbol{i} = 1 \ldots m, \forall \boldsymbol{j} = 1 \ldots m, \boldsymbol{i} \neq \boldsymbol{j}, \beta_j(p_i) > 0.$$

Se q, é ponto no interior do polígono, então $\beta_i(q) > 0$ para todo $\boldsymbol{i} \in 1 \ldots m$. Com isso é possível definir um polígono convexo, $\boldsymbol{\mathcal{P}}$, como:

$$\boldsymbol{\mathcal{P}} = \{q \in \mathbf{R}^2 \mid \forall \boldsymbol{i} = 1 \ldots m, \beta_i(q) > 0\},$$

Assumindo que os m segmentos de reta formem um polígono válido. Se $\beta_i(q) = 0$ para alguns, mas não todos, lados do polígono, q está localizado na fronteira do polígono.

Define-se q_β e β_{max} como sendo:

$$q_\beta = \arg\max \prod_{i=1}^{m} \beta_i(q) \qquad \text{(Eq. 3.2)}$$

$$\beta_{max} = \prod_{i=1}^{m} \beta_i(q_\beta) \qquad \text{(Eq. 3.3)}$$

Assim β_{max} é o valor máximo do produto das distancias para cada lado no interior do polígono. Define-se a função do produto dimensionado das distancias $\beta(q)$ como:

$$\beta(q) = \beta_{max}^{\frac{1-m}{m}} \prod_{i=1}^{m} \beta_i(q) \qquad \text{(Eq. 3.4)}$$

Lema[2]: O conjunto de máximos locais de $\beta(q)$ no interior do polígono $\boldsymbol{\mathcal{P}}$ é uma singularidade. Além disso, $\beta(q)$ é livre de mínimos locais no interior de $\boldsymbol{\mathcal{P}}$.

A partir do Lema, conclui-se que $\beta(q)$ decresce monotonicamente a medida que q se aproxima da fronteira de $\boldsymbol{\mathcal{P}}$ ao longo de um raio de q_β em todas as direções. Dado um polígono convexo válido, constrói-se o mapeamento para o disco usando $\beta(q)$.

Nota-se que as propriedades desejadas da função de navegação são invariantes a transformações do tipo translação e rotação. Sendo assim, primeiro aplica-se uma translação do polígono tornando o ponto q_β na origem (q_β = {0,0} no plano Euclidiano). Depois aplica-se uma rotação para que o ponto médio $\mathfrak{p}$ da face de saída do robô esteja no eixo negativo de $\boldsymbol{x}$ (para o caso da célula ser a célula que possui o ponto objetivo do robô, não é necessário aplicar esta rotação).

Dado um polígono convexo válido $\boldsymbol{\mathcal{P}}$ e a transformação $\boldsymbol{\mathcal{T}}$ (explicada anteriormente), define-se $\boldsymbol{\varphi : \mathcal{T}(\mathcal{P}) \rightarrow \mathcal{B}}$ Como sendo o mapeamento difeomórfico de um polígono convexo para o disco:

$$\varphi(q) = \frac{q}{||q|| + \beta(q)} \qquad \text{(Eq. 3.5)}$$

A Figura 7 mostra um mapeamento $\varphi(q)$ em um polígono convexo.

[2] A prova desse lema não será mostrada aqui, para mais informações consulte Coner (2007).

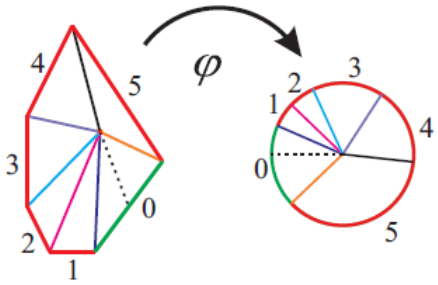

Figura 7: φ sendo aplicada em um polígono convexo.

O mapeamento φ é suficiente para se calcular as funções de navegação em células diferentes da célula final (célula que possui o ponto que corresponde a posição final do robô), porém para a célula final ainda é preciso que seja realizado mais um mapeamento, este por sua vez irá mapear o disco encontrado por φ para outro disco com o ponto que corresponde a posição final do robô na origem do disco. Para isso um mapeamento baseado em números complexos é suficiente, sendo $\boldsymbol{z} = q^b = \varphi(q)$ um ponto arbitrário no interior do disco representando no plano complexo e $\boldsymbol{z_g} = q_g^b = \varphi(q_g)$ sendo o ponto de destino do robô no plano complexo. Então, a função $\psi: B|_{q_g^b} \rightarrow B|_0$, é definida como:

$$\psi(z) = \frac{z - z_g}{1 - \overline{z_g} \cdot z} \qquad \text{(Eq. 3.6)}$$

A Figura 8 mostra o mapeamento ψ sendo aplicado a um disco, para o qual já foi calculada sua respectiva função de navegação.

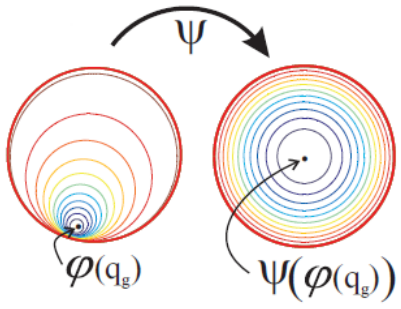

Figura 8: exemplo de mapeamento ψ.

Com o mapeamento pronto, resta apenas calcular a função de navegação para cada tipo de célula.

3.2. Função de navegação na célula

A primeira etapa para se calcular a função de navegação no disco é mudar de variáveis cartesianas para variáveis polares, portanto para $q_d = \varphi(q) = (x_d, y_d)$ define-se:

$$\rho = \sqrt{x_d^2 + y_d^2} \qquad \text{(Eq. 3.7)}$$

$$\theta = atan2(y_d, x_d) \qquad \text{(Eq. 3.8)}$$

O cálculo da função de navegação é diferente dependendo do tipo da célula em que está sendo feito o cálculo da função de navegação. Para as células livres (células que o robô tem com objetivo atravessar um dos lados do polígono), tem-se a seguinte fórmula para o cálculo da função de navegação:

$$\gamma_b(\rho, \theta) = \frac{\alpha_1 - \alpha_0}{2\pi} + \frac{1}{\pi} tan^{-1}\left(\frac{\rho\ \sin(\alpha_1 - \theta)}{1 - \rho\ \cos(\alpha_1 - \theta)}\right) - \frac{1}{\pi} tan^{-1}\left(\frac{\rho\ \sin(\alpha_0 - \theta)}{1 - \rho\ \cos(\alpha_0 - \theta)}\right) \qquad \text{(Eq. 3.9)}$$

Onde α_i representa as coordenadas do ângulo dos vértices da face pela qual o robô irá passar para a célula adjacente. A Figura 9 mostra o mapeamento φ e a condição de fronteira usada para o cálculo da função de navegação.

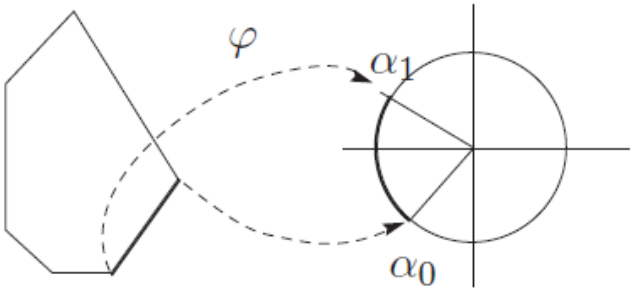

Figura 9: Mapeando polígono para o disco, com a face de saída identifica.

Para a célula final, a função de navegação é definida da seguinte maneira:

$$\gamma_g(q) = \frac{1}{2}\|\psi(\varphi(q))\|^2 \text{, onde } 0 \leq \gamma_g \leq 1. \qquad \textit{(Eq. 3.10)}$$

Com γ_b e γ_g é possível calcular a função de navegação para todo o espaço de configuração livre. Para as funções de navegação das células obstáculo do ambiente, basta atribuir um valor constante e muito superior aos valores encontrados para as demais células. Assim tem-se um valor potencial bem definido para cada ponto do ambiente.

3.3. Método implementado

A seguir o fluxograma (Figura 10) mostra os passos para calcular a função de navegação para uma célula. Os blocos marcados com asterisco serão detalhados mais adiante.

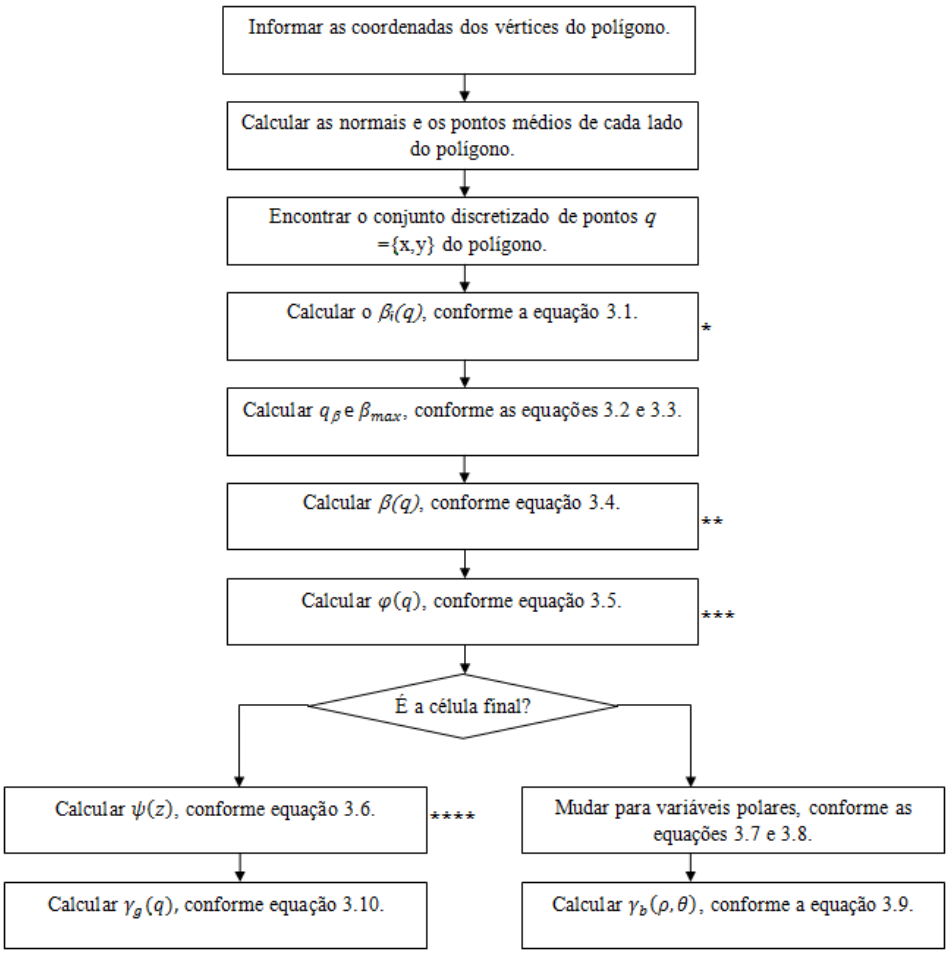

Figura 10: fluxograma da função de navegação de uma célula.

***** Para o pseudocódigo de $\beta_i(q)$ são necessárias as seguintes definições:

- $\mathbf{n}_i$: normal do *i*-ésimo lado do polígono convexo;
- $\mathbf{p}_i$: ponto médio do *i*-ésimo lado do polígono convexo;
- q : representa os pontos q ={x,y} do polígono convexo.

Entradas: $\mathbf{n}_i$, $\mathbf{p}_i$, q = {x,y}

Saída: $\beta_i(q)$

```
para k = 1 até i faça
     beta_i(q,k) = -n(k) . ( q - p(k) );
fim para
```

****** Para o pseudocódigo de $\beta(q)$ são necessárias as seguintes definições:

- m : número de lados do polígono convexo;
- β_{max} : o produto dos β_i(q), para o ponto q que faz esse produto ser o maior valor possível.

Entradas: m, β_{max}, β_i(q)

Saída: β(q)

```
beta_prod(q) = 1;
para x = 1 até i faça
     beta_prod(q) = beta_prod(q) * beta(q,i);
fim para
beta(q) = ((beta_max)^(1-m/m))*beta_prod(q);
```

******* Para o pseudocódigo de $\varphi(q)$ não é necessário acrescentar novas definições.

Entradas: q = {x,y}, β(q)

Saídas: φ(q) = {x_d,y_d}

```
x_d = phi(x) = x / ( (x^2 + y^2)^0.5 + beta(q) );
y_d = phi(y) = y / ( (x^2 + y^2)^0.5 + beta(q) );
```

******** Para o pseudocódigo de $\psi(z)$ são necessárias as seguintes definições:

- z : é um número complexo na forma $x_d + y_d j$;
- z_obj: ponto de destino do robô representando no plano complexo;
- ž_obj é o conjugado complexo de z_obj.

Entradas: φ(q) = {x_d,y_d}

Saídas: ψ(z)

```
z = x_d + y_d*j;
psi(z) = (z - z_obj) / (1 - ž_obj*z);
```

4. Resultados

O primeiro passo para avaliar a implementação do algoritmo de funções de navegação utilizado em Conner (2007) foi criar um ambiente para o robô poder navegar por ele. Este ambiente foi representado na Figura 11, onde é possível ver que este é composto por seis células "livres" e duas células "obstáculos". Com isso escolheu-se duas trajetórias distintas, com diferentes células iniciais, porém com a célula final idêntica. Foi escolhida a célula final igual para ambas a trajetórias por conveniência, pois dessa forma diminui o número de cálculos feitos para se obter duas trajetórias no mesmo ambiente. Deve-se notar também, que pelo fato de terem sido escolhidas duas trajetórias diferentes, nos segmentos {(1,1),(5,8)} e {(10,8),(15,6)} existem descontinuidades na função de navegação global (função de navegação de todo o ambiente).

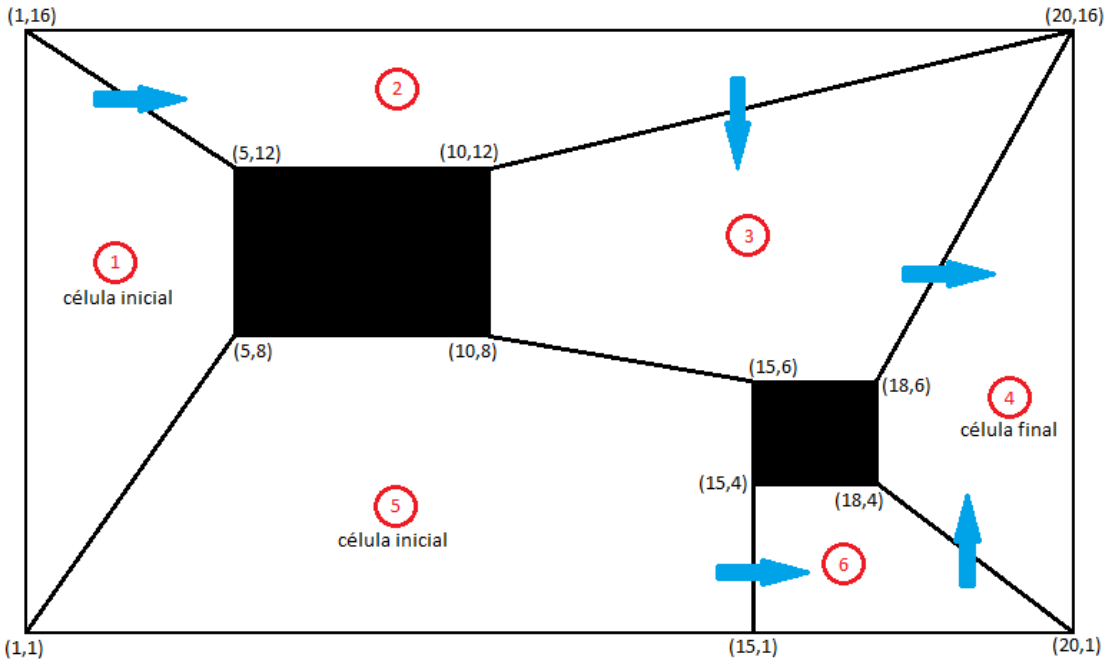

Figura 11: Ambiente de navegação criado para este trabalho.

Com o ambiente pronto**,** o próximo passo foi calcular a função de navegação em cada célula livre do ambiente, e atribuir um valor constante a função de navegação de cada célula obstáculo que seja maior que os valores encontrados para as funções potenciais das

células livres. O objetivo do conjunto de ferramentas desenvolvidos nessa etapa do trabalho foi combinar as funções de navegação de cada célula do ambiente proposto em uma matriz potencial. Para atingir esse objetivo foi codificado duas *functions* e um *script* (no Apêndice A) no sotfware Matlab®. Uma *function* exclusiva para o cálculo da função de navegação de células livres que é diferente da função de navegação da célula final, e a outra *funciton* apenas para o cálculo da função de navegação da célula final. E por fim o *script* (que invoca ambas as *funcitons)* atribui à função de navegação constante para as células obstáculos e exporta a matriz potencial em um arquivo texto.

4.1. *Function betaq_m*

A *function* que realiza o cálculo da função de navegação das células livres recebeu o nome de *betaq_m*. Esta *function* tem como saída a matriz potencial da célula livre e recebe como argumentos: o número de lados do polígono convexo (célula livre); o lado pelo o qual o robô atravessará para a outra célula; o vetor das coordenadas *x* dos vértices do polígono; o vetor das coordenadas *y* dos vértices do polígono; e por último o "grau" do polígono.

Abaixo um exemplo de chamada da *function betaq_m* (Figura 12).

```
>> vet_x = [3 5 4 2 1];
>> vet_y = [7 2 1 1 2];
>> A = betaq_m(5, 4, vet_x, vet_y, 2);
```

Figura 12: Chamada *function betaq_m*

Para *betaq_m* funcionar corretamente a ordem utilizada para declarar os $\{x_1, x_2,..., x_n\}$ de *vet_x* deve ser a mesma na hora de declarar os $\{y_1, y_2, ..., y_n\}$ de *vet_y*. O último argumento dessa função define o intervalo de valores da função potencial. No caso do exemplo mostrado na Figura 12, o intervalo é igual a 2, o que significa que o maior valor da função potencial calculada será igual 2 e o menor valor será igual 1.

Por convenção, foi definido que a numeração dos lados do polígono será no sentido horário, como é mostrado na Figura 13.

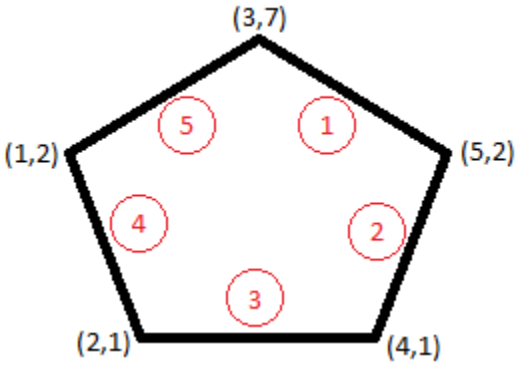

Figura 13: Numeração dos lados do polígono no sentindo horário

Para testar o código da *function betaq_m* desenvolvido, foi utilizada a ferramenta *plot* do Matlab® para desenhar o gráfico em 3D da função de navegação calculada para a célula livre em questão (pentágono utilizado na Figura 13). Abaixo estão duas figuras mostrando a função de navegação calculada (Figuras 14 e 15). Foi escolhido o 5° lado do pentágono como lado que o robô irá atravessar para a célula adjacente.

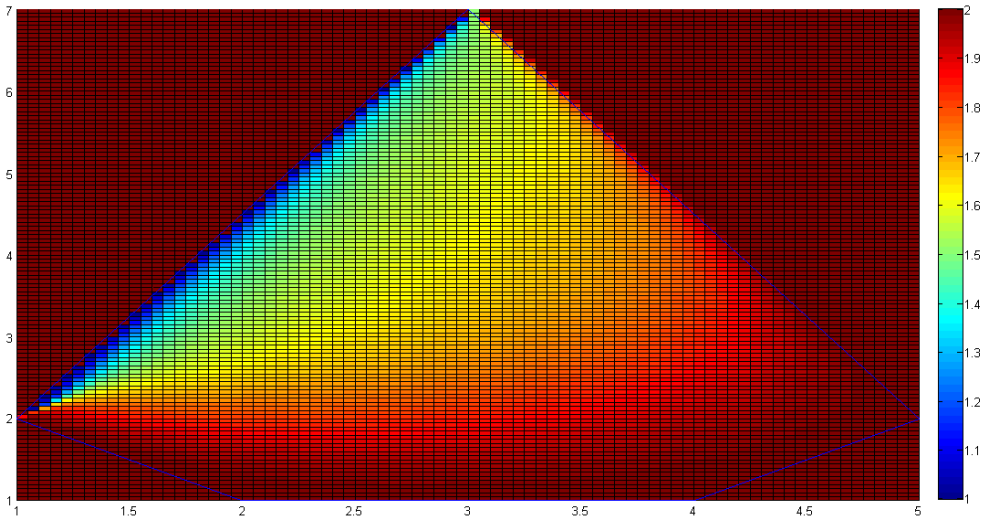

Figura 14: Vista superior de *betaq_m*

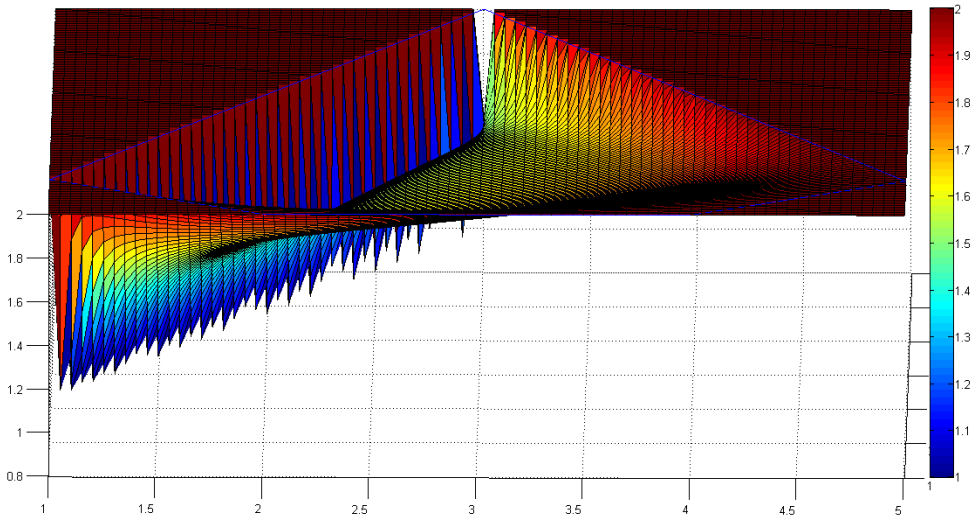

Figura 15: *betaq_m* vista 3D

4.2. *Funtion betaq_g*

A *function* que realiza o cálculo da função potencial da célula final, recebeu o nome de *betaq_g*. Ela é bastante semelhante à *function betaq_m*, no entanto, como o robô não sairá dessa célula, não é necessário passar o número do lado pelo qual o robô deve passar para a célula seguinte, e nem o grau do polígono, pois essa *function* já foi implementada levando em conta que o ponto de destino final do robô deve ser onde tem o menor valor da função de navegação (análogo ao polígono possuir grau 1).

Portanto os argumentos que *betaq_g* recebe são: o número de lados do polígono; o ponto de destino final do robô; o vetor das coordenadas *x* dos vértices do polígono; e o vetor das coordenadas *y* dos vértices do polígono**.**

Igual a *betaq_m*, *betaq_g* tem como saída à matriz potencial da célula final. A Figura 16 mostra um como utilizar a *function betaq_g* de maneira correta.

```
>> vet_x = [3 5 4 2 1];
>> vet_y = [7 2 1 1 2];
>> B = betaq_g(5, [3 4], vet_x, vet_y);
```

Figura 16: Exemplo *function betaq_g*

Para *betaq_g* funcionar corretamente o segundo argumento deve ser da seguinte forma [x_f y_f]. Os resultados obtidos para a célula final são expostos nas Figuras 17 e 18.

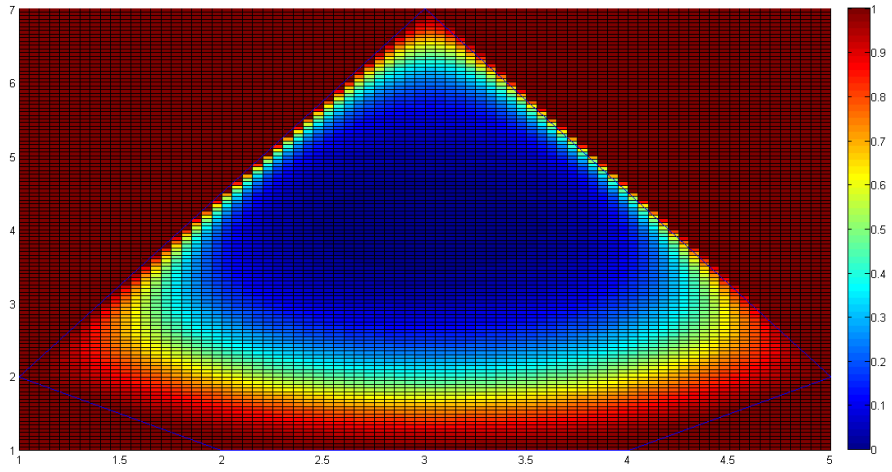

Figura 17: Vista superior de *betaq_g*

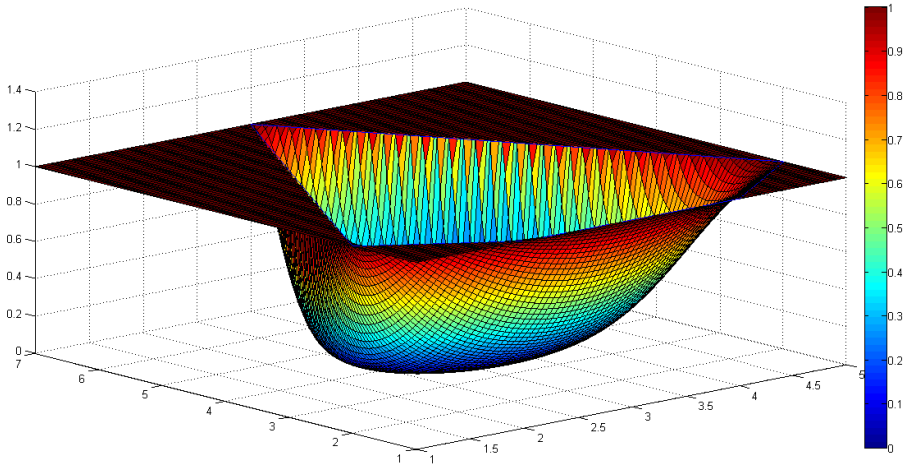

Figura 18: *betaq_g* vista 3D

4.3. *Script Calc_amb*

Finalmente ao *script* foi dado o nome *calc_amb*. Como já foi mencionado acima o *script* apenas invoca as *functions* e organiza a matriz potencial do ambiente. O resultado da execução de *calc_amb* no ambiente proposto é uma matriz potencial de 301x381, sendo as coordenadas de *y* do mapa do ambiente representadas pelas linhas e a coordenadas *x* pelas colunas. A Figura 19 abaixo mostra uma imagem da função de navegação global (matriz potencial) calculada para o ambiente proposto.

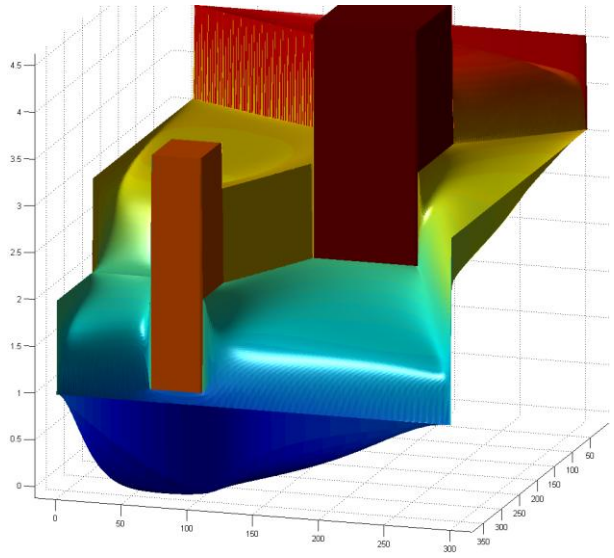

Figura 19: Função de navegação global.

4.4. Gradiente da função de navegação

Com o conjunto de ferramentas desenvolvido (*script* + *functions*), calcula-se a função de navegação de um ambiente completo ou de apenas uma célula (utilizando apenas uma das duas *functions*). O robô se movimentará seguindo o gradiente negativo da função de navegação. Na Figura 20, mostra-se um exemplo do gradiente negativo de um pentágono (que representa uma célula), e com o auxílio de setas, mostra-se a direção que o robô deverá seguir.

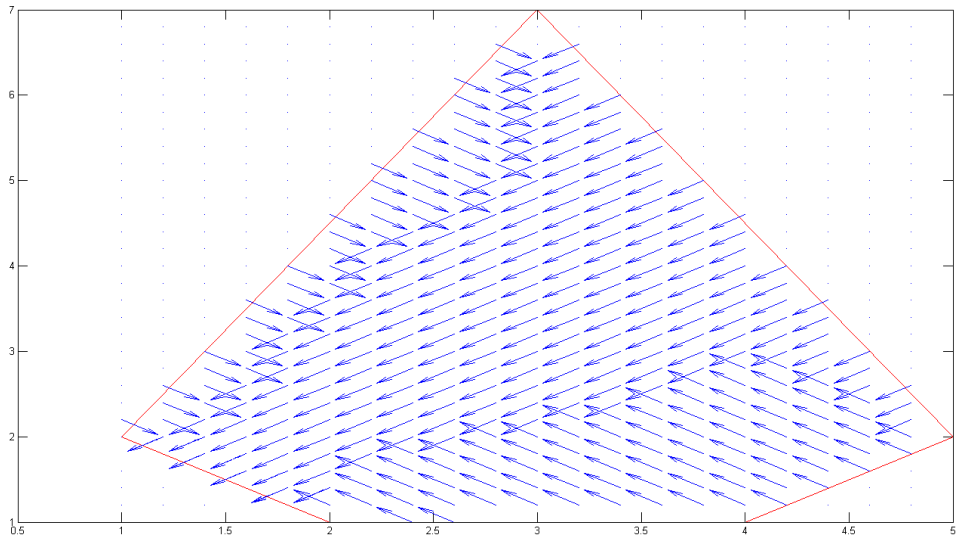

Figura 20: Gradiente negativo da função de navegação.

5. Conclusão

Através do desenvolvimento deste trabalho de conclusão de curso, foi possível estudar uma variedade de teorias que são utilizadas para solucionar problemas de movimentação e navegação de robôs autônomos. Dentre essas teorias estudadas, foi escolhido o método de seqüência de função de navegação para ser implementado, pois abrange dois outros métodos muito conhecidos em robótica: decomposição do ambiente em células e função de navegação. Com essa implementação, ressaltou-se a possibilidade de utilização desse método para controlar robôs.

Outro fator que ressalta-se com o desenvolvimento desse trabalho, é que, dentre as teorias de campos potenciais, o método implementado se destacou, pois como o ambiente é dividido em regiões menores, denominadas células, com geometria simples (polígonos convexos para o $\mathbf{R}^2$, poliedros convexos para o $\mathbf{R}^3$), é mais fácil, matematicamente, definir uma função de navegação. Sendo assim o método torna-se menos custoso computacionalmente. Contudo, nota-se que para se utilizar o método escolhido é necessário possuir um conhecimento prévio do ambiente, o que é uma restrição para muitos problemas de robótica. Além disso, para a utilização do algoritmo implementado, este trabalho de conclusão de curso teve que considerar que o ambiente seja estático, ou seja, o ambiente sempre será da forma como foi considerado inicialmente. Com isso, caso o ambiente mude, por exemplo, um obstáculo novo é acrescentado ao ambiente, pode-se utilizar o resultado encontrado pelo algoritmo implementado em conjunto com informações obtidas por sensores do robô de forma reativa. O controle de robôs que possuem comportamentos reativos é explicado detalhadamente em Grassi Junior (2006).

Nesse trabalho, foi utilizado o Matlab® para implementar o algoritmo proposto por Conner (2007), pois esse software já possui diversas funções de visualização prontas (*plot*, *plot3*, *surf*, *contour*, entre outras), o que facilitou a interpretação dos resultados obtidos. Esse fator foi de extrema importância para o presente trabalho, pois, com a visualização gráfica dos resultados, verificou-se que a implementação está de acordo com o esperado, o que, se fosse implementado na linguagem do C/C++ haveria a ausência da visualização, o que seria um obstáculo. No entanto, analisando de maneira computacional, a implementação em Matlab® é um barreira para uma solução que utilize o algoritmo, mas se

traduzido para a linguagem de programação C/C++, computacionalmente, a implementação será menos custoso para o computador/hardware utilizado. Por isso, uma melhoria proposta para esse trabalho de conclusão de curso é traduzir o algoritmo em linguagem C/C++.

Esse trabalho contribui também com informações sobre um ambiente previamente conhecido, o que pode auxiliar o controle de um robô que está confinado a esse ambiente. Sendo assim, uma importante ferramenta para o desenvolvimento de softwares de controle baseados em funções de navegação para robôs móveis. Com isso, uma sugestão de continuidade, seria desenvolver softwares que utilizem a matriz de saída para controlar robôs em diferentes ambientes.

Outra sugestão de continuidade deste trabalho de conclusão de curso é, a partir do resultado obtido, calcular um campo vetorial que leve em consideração a dinâmica e a cinética do robô a ser utilizado, assim introduzindo teoria de controle no algoritmo implementado no presente trabalho.

6. Referências

Choset, H.; Lynch, K. M.; Hutchinson, S.; Kantor, G.; Burgard, W.; Kavraki, L. E.; and Thrun, S. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, Boston, 2005.

Conner, D. C. *Integrating Planning and Control for Constrained Dynamical Systems.* Thesis (Doctor of Philosophy in Robotics). University of Pennsylvania. 2007. 238p.

Conner, D. C.; Rizzi, A. A.; and Choset, H. Composition of local potential functions for global robot control and navigation. In *Proceedings of the 2003 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems (IROS)*, pages 3546-3551, Las Vegas, Nevada, October 2003.

Grassi Junior, V. *Arquitetura Híbrida para Robôs Móveis Baseada em Funções de Navegação com Interação Humana*. Tese de Doutorado em Engenharia. Universidade de São Paulo. 2006. 120p.

Latombe, J. *Robot Motion Planning*. Kluwer, Boston, 1991.

LaValle, S. M. *Planning Algorithms*. Cambridge University Press, 2006.

Rimon, E.; Koditschek, D. E. Exact robot navigation using artificial potential functions. *IEEE Trans. on Robotics and Automation*, v. 8,n. 5,p. 501-518, 1992.

7. Apêndice A

function betaq_m:

```
function [pot_M] = betaq_m(tam, outlet, p_x, p_y, grau)

incx = 0.05;
incy = 0.05;

if (outlet > tam)
      error('Outlet zone inválida!!!')
end

px = [p_x p_x(1)];
py = [p_y p_y(1)];

x_out = [px(outlet) px(outlet+1)];
y_out = [py(outlet) py(outlet+1)];

out = [x_out' y_out'];
out = out';

% Calcula os pontos médios
%

for i = 2:tam+1
      p_medio_x(i-1) = (px(i-1) + px(i))/2;
      p_medio_y(i-1) = (py(i-1) + py(i))/2;
end
ponto_m = [p_medio_x' p_medio_y'];

% Calcula as normais
%

for i = 2:tam+1
      normal_x(i-1) = px(i) - px(i-1);
      normal_y(i-1) = py(i-1) - py(i);
end
normal = [normal_y' normal_x'];

% ------------------
% Cálculo do vetor Q
% ------------------

% O vetor Q representa os pontos q's utilizados na fórmula do beta_i;
%

q_x = [min(p_x):incx:max(p_x)];
q_y = [min(p_y):incy:max(p_y)];
```

```
cont = 1;
for x = 1:length(q_x)
    for y = 1:length(q_y)
            Q(:,cont) = ([q_x(x);q_y(y)]);
            if (q_x(x) == out(1,1) & q_y(y) == out(2,1))
                  out_vert(1) = cont;
            elseif (q_x(x) == out(1,2) & q_y(y) == out(2,2))
                  out_vert(2) = cont;
            end
            for i = 1:tam
                  if (q_x(x) == p_x(1,i) & q_y(y) == p_y(1,i))
                        ind_vert(i) = cont;
                  end
            end
            cont = cont+1;
      end
end

% -----------------
% Cálculo do beta_i
% -----------------

% beta_i(q) = -n_i . (q - p_i)
%

ponto_m = ponto_m';
normal = normal';

for i = 1:tam
      m = ones(size(Q));
      m(1,:) = ponto_m(1,i).*m(1,:);
      m(2,:) = ponto_m(2,i).*m(2,:);

      aux_z(:,:,i) = (Q - m);

      n = ones(size(Q));
      n(1,:) = normal(1,i).*n(1,:);
      n(2,:) = normal(2,i).*n(2,:);

      beta_i(:,i) = -dot(n,aux_z(:,:,i));
end

beta_i = beta_i';

% ---------------
% Cálculo do beta
% ---------------

% beta_q(q) = beta_max^k * Prod(beta_i(q))
%

beta = prod(beta_i);
beta = beta';

contador_positivo = 0;
contador_zero = 0;
```

```
for i=1:length(beta_i)
      if (beta(i) >= 0)
            for j=1:tam
                  if (beta_i(j,i) > 0)
                        contador_positivo = contador_positivo+1;
                  elseif (beta_i(j,i) == 0)
                        contador_zero = contador_zero+1;
                  end
            end
            if ((contador_positivo+contador_zero == tam) & (contador_zero
~= tam))
                  beta_t(i) = 1;
            else
                  beta_t(i) = 0;
            end
      else
            beta_t(i) = 0;
      end
      contador_positivo = 0;
      contador_zero = 0;
end

beta_t = beta_t';
[beta_max,q_b] = max(beta);
beta = (beta_max^((1-tam)/tam))*beta;
beta_mask = (beta_t > 0);

% --------------
% Cálculo do phi
% --------------

beta = beta';

phi(1,:) = Q(1,:)./( (Q(1,:).^2 + Q(2,:).^2).^0.5 + beta );
phi(2,:) = Q(2,:)./( (Q(1,:).^2 + Q(2,:).^2).^0.5 + beta );

% Translação
%

Q_T = ones(size(Q));
Q_T(1,:) = Q_T(1,:) * Q(1,q_b);
Q_T(2,:) = Q_T(2,:) * Q(2,q_b);
Q_T = Q - Q_T;

% Rotação
%

alpha_rot = atan2(normal(2,outlet),normal(1,outlet));
omega = pi - alpha_rot;
Rot = [cos(omega) -sin(omega); sin(omega) cos(omega)];

Q_TR = Rot * Q_T;
Q_T = Q_TR;

phi_T(1,:) = Q_T(1,:)./( (Q_T(1,:).^2 + Q_T(2,:).^2).^0.5 + beta ) ;
phi_T(2,:) = Q_T(2,:)./( (Q_T(1,:).^2 + Q_T(2,:).^2).^0.5 + beta ) ;
phi = phi_T;
```

```
alpha(:,1) = phi(:,out_vert(1));
alpha(:,2) = phi(:,out_vert(2));

[alpha1_ang,alpha1_ro] = cart2pol(alpha(1,1),alpha(2,1));
[alpha2_ang,alpha2_ro] = cart2pol(alpha(1,2),alpha(2,2));

% Testa e reorganiza os alpha0 e alpha1
%

if (alpha1_ang < 0)
    alpha1_ang = alpha1_ang+2*pi;
end
if (alpha2_ang < 0)
   alpha2_ang = alpha2_ang+2*pi;
end

if (alpha2_ang > alpha1_ang)
      lixo = alpha1_ang;
      alpha1_ang = alpha2_ang;
      alpha2_ang = lixo;
end

clear lixo;

% Passa para coordenada Polar
%

[theta, rho] = cart2pol(phi(1,:), phi(2,:));

% Calcula a função potencial
%

pot = (alpha2_ang - alpha1_ang) ./ (2 * pi) + (pi^-1) * atan((rho .*
sin(alpha2_ang*ones(size(theta)) - theta)) ./ (1 - rho .*
cos(alpha2_ang*ones(size(theta)) - theta))) - (pi^-1) * atan((rho .*
sin(alpha1_ang*ones(size(theta)) - theta)) ./ (1 - rho .*
cos(alpha1_ang*ones(size(theta)) - theta)));

for i=1:length(pot)
      if (beta_mask(i) > 0)
            pot(i) = pot(i)*(beta_mask(i)')+grau;
      else
            pot(i) = 10;
      end
end

% Reorganiza o vetor potencial em uma matriz
%

dx = max(size(q_x));
dy = max(size(q_y));
pot_M = [pot(1:dy)'];
for i=1:(dx-1)
    pot_M = [pot_M pot(i*dy + 1:(i+1)*dy)'];
end
```

function betaq_g:

```
function [pot_M] = betaq_g(tam, goal, p_x, p_y)

incx = 0.05;
incy = 0.05;

px = [p_x p_x(1)];
py = [p_y p_y(1)];

% Calcula os pontos médios
%

for i = 2:tam+1
      p_medio_x(i-1) = (px(i-1) + px(i))/2;
      p_medio_y(i-1) = (py(i-1) + py(i))/2;
end
ponto_m = [p_medio_x' p_medio_y'];

% Calcula as normais
%

for i = 2:tam+1
      normal_x(i-1) = px(i) - px(i-1);
      normal_y(i-1) = py(i-1) - py(i);
end
normal = [normal_y' normal_x'];

% ------------------
% Cálculo do vetor Q
% ------------------

% O vetor Q representa os pontos q's utilizados na fórmula do beta_i;
%

q_x = [min(p_x):incx:max(p_x)];
q_y = [min(p_y):incy:max(p_y)];

cont = 1;
for x = 1:length(q_x)
    for y = 1:length(q_y)
            Q(:,cont) = ([q_x(x);q_y(y)]);
            if ([q_x(x) q_y(y)] == goal)
                  goal_vert = cont;
            end
            cont = cont+1;
      end
end

% -----------------
% Cálculo do beta_i
% -----------------

% beta_i(q) = -n_i . (q - p_i)
%
```

```
ponto_m = ponto_m';
normal = normal';

for i = 1:tam
      m = ones(size(Q));
      m(1,:) = ponto_m(1,i).*m(1,:);
      m(2,:) = ponto_m(2,i).*m(2,:);

      aux_z(:,:,i) = (Q - m);

      n = ones(size(Q));
      n(1,:) = normal(1,i).*n(1,:);
      n(2,:) = normal(2,i).*n(2,:);

      beta_i(:,i) = -dot(n,aux_z(:,:,i));
end
beta_i = beta_i';

% ---------------
% Cálculo do beta
% ---------------

beta = prod(beta_i);
beta = beta';

contador_positivo = 0;
contador_zero = 0;

for i=1:length(beta_i)
      for j=1:tam
            if (beta_i(j,i) >= 0)
                  contador_positivo = contador_positivo+1;
            elseif (beta_i(j,i) == 0)
                  contador_zero = contador_zero+1;
            end
      end
      if ((contador_positivo == tam) & (contador_zero ~= tam))
            beta_t(i) = 1;
      else
            beta_t(i) = 0;
      end
      contador_positivo = 0;
      contador_zero = 0;
end

beta_t = beta_t';
[beta_max,q_b] = max(beta);
beta = (beta_max^((1-tam)/tam))*beta;
beta_mask = (beta_t > 0);

% --------------
% Cálculo do phi
% --------------

beta = beta';
```

```
phi(1,:) = Q(1,:)./( (Q(1,:).^2 + Q(2,:).^2).^0.5 + beta );
phi(2,:) = Q(2,:)./( (Q(1,:).^2 + Q(2,:).^2).^0.5 + beta );

% Translação
%

Q_T = ones(size(Q));
Q_T(1,:) = Q_T(1,:) * Q(1,q_b);
Q_T(2,:) = Q_T(2,:) * Q(2,q_b);
Q_T = Q - Q_T;

phi_T(1,:) = Q_T(1,:)./( (Q_T(1,:).^2 + Q_T(2,:).^2).^0.5 + beta ) ;
phi_T(2,:) = Q_T(2,:)./( (Q_T(1,:).^2 + Q_T(2,:).^2).^0.5 + beta ) ;
phi = phi_T;

% Aplica o mapeamento no plano complexo para ajeitar o campo potencial
%
% zeta = (z - z_g) / (1 - z_g_conjugado * z)
%

z = complex(phi(1,:),phi(2,:));
zeta = ((z - z(goal_vert))./(1 - conj(z(goal_vert)).*z));
pot = (abs(zeta)).^2;

for i=1:length(pot)
      if (beta_mask(i)' == 0)
            pot(i) = 5;
      else
            pot(i) * beta_mask(i)';
      end
end

% Reorganiza o vetor potencial em uma matriz

dx = max(size(q_x));
dy = max(size(q_y));
pot_M = [pot(1:dy)'];
for i=1:(dx-1)
    pot_M = [pot_M pot(i*dy+1:(i+1)*dy)'];
end
```

script calc_amb:

```
clear all;
close all;

% Incremento
%

inc = 0.05;
tamanho = [301 381];

% Trajetória 1
%

pot1 = betaq_m(4,1,[1 5 5 1],[16 12 8 1],4);
pot2 = betaq_m(4,2,[1 20 10 5],[16 16 12 12],3);
pot3 = betaq_m(5,1,[20 18 15 10 10],[16 6 6 8 12],2);

% Trajetória 2
%

pot5 = betaq_m(6,4,[5 10 15 15 15 1],[8 8 6 4 1 1],3);
pot6 = betaq_m(4,2,[15,18,20,15],[4 4 1 1],2);

% Célula Final
%

pot4 = betaq_g(4,[19 5],[20 20 18 18],[16 1 4 6]);

% Células Obstáculos
%

cel1 = 8*ones(length([5:inc:10]),length([8:inc:12]));
cel2 = 8*ones(length([15:inc:18]),length([4:inc:6]));

% Inicializando Matriz Potencial
%

Pot_final = 10*ones(length(1:inc:16),length(1:inc:20));

% Combinando as funções potenciais
%

for i=1:max(size(pot1))
      for j=1:min(size(pot1))
            if pot1(i,j) < Pot_final(i,j)
                  Pot_final(i,j) = pot1(i,j);
            end
      end
end
```

```
for i=1:min(size(pot2))
      for j=1:max(size(pot2))
            if pot2(i,j) < Pot_final(i+220,j)
                  Pot_final(i+220,j) = pot2(i,j);
            end
      end
end

for i=1:max(size(cel1))
      for j=1:min(size(cel1))
            if cel1(i,j) < Pot_final(j+140,i+80)
                  Pot_final(j+140,i+80) = cel1(i,j);
            end
      end
end

for i=1:max(size(pot3))
      for j=1:min(size(pot3))
            if pot3(i,j) < Pot_final(i+100,j+180)
                  Pot_final(i+100,j+180) = pot3(i,j);
            end
      end
end

for i=1:min(size(pot5))
      for j=1:max(size(pot5))
            if pot5(i,j) < Pot_final(i,j)
                  Pot_final(i,j) = pot5(i,j);
            end
      end
end

for i=1:max(size(cel2))
      for j=1:min(size(cel2))
            if cel2(i,j) < Pot_final(j+60,i+280)
                  Pot_final(j+60,i+280) = cel2(i,j);
            end
      end
end

for i=1:min(size(pot6))
      for j=1:max(size(pot6))
            if pot6(i,j) < Pot_final(i,j+280)
                  Pot_final(i,j+280) = pot6(i,j);
            end
      end
end

for i=1:max(size(pot4))
      for j=1:min(size(pot4))
            if pot4(i,j) <= Pot_final(i,j+340)
                  Pot_final(i,j+340) = pot4(i,j);
            end
      end
end
save -ascii 'Matriz Potencial.txt' tamanho;
save -ascii -append 'Matriz Potencial.txt' Pot_final;
```