Universidade Federal
do Rio de Janeiro
Escola Politécnica

ESTRATÉGIA DE DESVIO DE OBSTÁCULOS E PLANEJAMENTO DE TRAJETÓRIA PARA UMA CADEIRA DE RODAS AUTÔNOMA

RODOLPHO COSTA RIBEIRO

Projeto de Graduação apresentado ao Curso de Engenharia de Controle e Automação da Escola Politécnica, Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Engenheiro.

Orientador:
Prof. Fernando Cesar Lizarralde, D.Sc.

Rio de Janeiro, RJ – Brasil

Março de 2015

ESTRATÉGIA DE DESVIO DE OBSTÁCULOS E PLANEJAMENTO DE TRAJETÓRIA PARA UMA CADEIRA DE RODAS AUTÔNOMA

RODOLPHO COSTA RIBEIRO

PROJETO DE GRADUAÇÃO SUBMETIDO AO CORPO DOCENTE DO CURSO DE ENGENHARIA DE CONTROLE E AUTOMAÇÃO DA ESCOLA POLITÉCNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO DE CONTROLE E AUTOMAÇÃO

Aprovado por:

_____

Prof. Fernando Cesar Lizarralde, D.Sc.

_____

Dr. Antonio Candea Leite, D.Sc.

_____

Prof. Alessandro Jacoud Peixoto, D.Sc.

Rio de Janeiro, RJ – Brasil

Março de 2015

Ribeiro, Rodolpho Costa

Estratégia de Desvio de Obstáculos e Planejamento de Trajetória para uma Cadeira de Rodas Autônoma / Rodolpho Costa Ribeiro. – Rio de Janeiro: UFRJ/ Escola Politécnica, 2015.

IX, 101 p. : il. ; 29,7 cm.

Orientador: Prof. Fernando Cesar Lizarralde, D.Sc.

Projeto de Graduação - UFRJ/ Escola Politécnica/ Curso de Engenharia de Controle e Automação, 2015.

Referências Bibliográficas: p.101.

1. Planejamento de Trajetória 2. Robótica Autônoma 3. Percepção I. Lizarralde, Fernando Cesar. II. Universidade Federal do Rio de Janeiro, Escola Politécnica, Curso de Engenharia de Controle e Automação. III. Estratégia de Desvio de Obstáculos e Planejamento de Trajetória para uma Cadeira de Rodas Autônoma.

Agradecimentos

Agradeço ao Gustavo Freitas por me emprestar a cadeira de rodas para esse trabalho.

Agradeço ao Aluizio Netto por me permitir usar o seu trabalho com comandos dados a partir de ondas cerebrais no meu trabalho.

Agradeço ao Professor Fernando Lizarralde pela orientação.

Resumo do Projeto de Graduação apresentado à Escola Politécnica/ UFRJ como parte dos requisitos necessários para a obtenção do grau de Engenheiro de Controle e Automação.

ESTRATÉGIA DE DESVIO DE OBSTÁCULOS E PLANEJAMENTO DE TRAJETÓRIA PARA UMA CADEIRA DE RODAS AUTÔNOMA

Rodolpho Costa Ribeiro

Março/2015

Orientador: Prof. Fernando Cesar Lizarralde, D.Sc.

Curso: Engenharia de Controle e Automação

Este trabalho apresenta uma estratégia de planejamento de trajetória para que um robô, partindo de qualquer posição inicial, atinja uma dada referência sem sofrer colisões com obstáculos estáticos ou móveis utilizando o Método dos Campos Potenciais Artificiais. Também é apresentada a implementação dessa estratégia em uma cadeira de rodas motorizada. A cadeira é um robô não-holonômico que detecta os obstáculos utilizando um *laser scanner*, cujos dados são enviados ao computador de bordo e recebe os comandos de velocidade de um microcontrolador Arduino. O *software* utilizado para a implementação é o *Robot Operating System* (ROS), um *framework cross-platform* que simplifica o desenvolvimento de sistemas robóticos e contém muitas bibliotecas úteis ao trabalho. Também é apresentada, como aplicação prática do planejamento de trajetória, uma tarefa de navegação em que a cadeira se move de forma autônoma por corredores, recebendo apenas comandos simples de virar à esquerda, à direita ou seguir em frente. Esses comandos podem ser dados por *joystick* ou por ondas cerebrais. Dessa forma, esse sistema de navegação autônoma pode ser usado por pessoas com deficiência física que utilizem a cadeira. Para essa parte do trabalho, é utilizado também um método de extração de linhas a partir dos dados do *laser scanner* conhecido como *Random Sample Consensus* (RANSAC) e o método de localização do robô PlICP.

Palavras-chave: Planejamento de Trajetória, Robótica Autônoma, Percepção.

Abstract of Undergraduate Project presented to POLI/UFRJ as a partial fulfillment of the requirements for the degree of Engineer of Control and Automation.

OBSTACLE AVOIDANCE AND PATH PLANNING STRATEGY FOR AN AUTONOMOUS WHEELCHAIR

Rodolpho Costa Ribeiro

March/2015

Advisor: Prof. Fernando Cesar Lizarralde, D.Sc.

Course: Enginnering of Control and Automation

This work presents a path planning strategy so that a robot, starting from any initial position, can reach a given reference avoiding static and mobile obstacles using the Artificial Potential Fields Method. It also presents an implementation for this strategy using an electric wheelchair. The wheelchair is a non-holonomic robot which detects obstacles using a laser scanner whose data is sent to an onboard computer and receives velocity inputs from an Arduino microcontroller. The software used for implementation is Robot Operating System (ROS), a cross-platform framework which simplifies the development of robotic systems and has many useful libraries for this work. It also presents, as a practical application for path planning, a navigation task where the wheelchair moves autonomously through a series of hallways, receiving simple inputs: go ahead, turn left or right and stop. These inputs are given either by joystick or by brain waves. Thus, this autonomous navigation system can be used by people with physical disabilities. For this part of the work, a line extraction method which uses the laser scanner data, known as Random Sample Consensus (RANSAC) and a robot localization method known as PlICP, will be used.

Keywords: Path Planning, Autonomous Robotics, Perception.

Índice:

Capítulo 1

Introdução

A robótica é um campo da tecnologia que surgiu da interseção entre diversas áreas, tais como Engenharias Eletrônica, Elétrica e Mecânica. Se a integração de tanta variedade de conhecimentos em sistemas robóticos os torna inerentemente complexos, por outro lado, garante aos robôs a flexibilidade necessária para que sejam utilizados em todo tipo de aplicação prática. Inicialmente, os robôs eram projetados para executar tarefas simples e repetitivas, tipicamente em fábricas trabalhando na manufatura de produtos. Nas últimas décadas, porém, tem havido uma mudança de foco para robôs capazes de executar tarefas mais complexas e que exigem do robô a habilidade de se adaptar a situações inesperadas [1]. Dessa forma surgiu a robótica autônoma.

Um robô autônomo é um robô capaz de executar tarefas com alto grau de independência. Um dos tópicos mais importantes da robótica autônoma é permitir que o robô interaja com o seu ambiente, seja na terra, ar, subsolo, no fundo do mar ou no espaço. As características básicas que tornam um robô autônomo são:

- Trabalhar por um longo período sem intervenção humana;
- Obter informações sobre seu ambiente;
- Se mover pelo ambiente sem assistência humana;
- Se adaptar às mudanças no ambiente, alterando a forma como executa suas tarefas;
- Evitar situações que provoquem danos às pessoas, às instalações ou ao ambiente de operação ou a si mesmo.

A capacidade de trabalhar sem intervenção humana depende principalmente do sistema energético do robô. Por outro lado, as demais características são englobadas na área de planejamento de trajetória, que é um dos maiores desafios em robótica autônoma atualmente.

Existem quatro objetivos básicos para o planejamento de trajetória, embora nem todos sejam necessários para toda aplicação em que o robô seja usado:

- a *navegação*, que consiste em encontrar um caminho para o robô, seja um braço robótico ou um robô móvel, partindo de uma certa configuração inicial para uma configuração final desejada evitando que qualquer ponto do robô esbarre em qualquer um dos obstáculos existentes no ambiente. Como o objetivo é garantir que nenhum ponto do robô toque em um obstáculo, é necessário usar uma representação matemática para o robô que englobe todos os seus pontos. Essa representação é chamada de configuração do robô;
- a *localização*, a capacidade do robô de determinar sua configuração atual com base nos dados obtidos pelos seus sensores. Localização é uma etapa essencial para a navegação precisa do robô em ambientes desconhecidos, já que ele precisa conhecer sua própria configuração para determinar a melhor forma de atingir a configuração final;
- a *cobertura*, o problema de escolher uma trajetória de forma que o robô passe por todos os pontos do espaço de configuração;
- o *mapeamento*, o problema de explorar um ambiente desconhecido e, através dos dados dos sensores, construir uma representação do seu mapa.

Posteriormente, objetivos opcionais passaram a ser considerados, como a otimização do tempo e do gasto energético do robô durante a trajetória. O presente trabalho, porém, trata apenas da navegação e localização do robô.

As características físicas do robô também são importantes, pois afetam a capacidade do algoritmo responsável pelo planejamento. Entre as mais importantes estão os graus de liberdade do robô, a forma do seu espaço de configuração e as restrições de velocidade a que o robô está sujeito. Tais características podem impedir que o robô se movimente na direção determinada pelo algoritmo em um certo instante e com isso ele pode ficar parado na posição atual. Tais características físicas são determinadas através do modelo do robô. O modelo escolhido também determina quais são os sinais de controle transmitidos ao robô: velocidades, para o modelo cinemático ou forças generalizadas (forças e torques) para o modelo dinâmico. O algoritmo de planejamento de trajetória utilizado foi o Método dos Campos Potenciais Artificiais, um método computacionalmente eficiente por não exigir cálculos complexos para determinar os sinais de controle enviados ao robô. O método foi originalmente descrito

em [2], motivado pela necessidade de um controle reativo (em que o sinal de controle depende de mudanças no ambiente) a ser usado com robôs operando em ambientes hostis e se tornou um dos mais populares na comunidade acadêmica, mesmo fora da área de robótica, embora esse método sofra do problema da existência de mínimos locais.

1.1 Motivação

A pesquisa na área de planejamento de trajetória tem se tornado cada vez mais crucial, pois as tarefas atuais que são determinadas para os robôs são menos repetitivas do que na manufatura industrial. Além disso, os ambientes de operação são menos estruturados, como transporte, cirurgias, exploração dos oceanos e do espaço, missões de busca e salvamento e robôs de assistência aos idosos e deficientes [1]. Para essas aplicações, é impossível programar o robô para cada situação específica. Além dessas aplicações em robótica, o planejamento de trajetória também já é utilizado em outras áreas como animação computacional, para determinar a movimentação de personagens em um *video game* ou em simuladores de equipamentos; programas de CAD, para verificar se uma peça ou equipamento pode ser montado; e determinação de rotas entre os componentes de circuitos integrados.

1.2 Objetivos

O planejamento de trajetória foi implementado em uma cadeira de rodas motorizada modelo Jazzy 600 de três formas diferentes:

- Controle manual da cadeira por um usuário através de um *joystick* analógico ou por ondas cerebrais. Nesse caso, o Método dos Campos Potenciais Artificiais é utilizado apenas para evitar colisões com obstáculos.
- Planejamento de trajetória completo. Nesse caso, o usuário apenas determina uma referência de configuração e a cadeira se desloca pelo ambiente de forma completamente autônoma até atingir a referência, sem atingir obstáculos.
- Deslocamento autônomo da cadeira por um conjunto de corredores através de comandos simples de seguir em frente, parar ou virar no próximo corredor à esquerda ou à direita. Esses comandos são dados pelos botões de um controle ou

por ondas cerebrais. Em uma situação real na qual um portador de deficiência física use a cadeira, ele pode não ser capaz de utilizar um *joystick* e deverá usar os comandos cerebrais. Estes, por sua vez, não podem ser complexos como os de um *joystick* analógico, para facilitar a adaptação do portador ao sistema.

Figura 1.1: Cadeira de rodas *Jazzy 600.*

1.3 Organização do Trabalho

Neste Capítulo 1 foi apresentada uma introdução à robótica autônoma e ao planejamento de trajetória. Também foram apresentados os objetivos desse trabalho.

No Capítulo 2 são definidos os conceitos e notações matemáticos utilizados ao longo do trabalho. Também é apresentada a modelagem cinemática para robôs com rodas e a análise de controlabilidade para estes mecanismos, descrevendo como as restrições cinemáticas produzidas pelas rodas afetam o deslocamento do robô e, portanto, os sinais de controle transmitidos a ele.

No Capítulo 3 é apresentado o Método dos Campos Potenciais Artificiais para a navegação do robô.

No Capítulo 4 é apresentado o algoritmo ICP e sua variante PlICp, que é utilizada para a localização do robô na implementação. Um algoritmo de localização é uma etapa necessária para a navegação autônoma usando o Método dos Campos Potenciais Artificiais.

No Capítulo 5 é apresentado o algoritmo RANSAC para extração de linhas a partir dos conjuntos de dados obtidos pelos sensores de um robô. Esse método é necessário para identificar os corredores pelos quais a cadeira se desloca.

No Capítulo 6 é apresentada a implementação de todos os algoritmos descritos nos capítulos anteriores. São descritos todos os equipamentos e *softwares* utilizados e o modo como eles se conectam. Também são descritos todos os programas criados pelo autor para esse trabalho. Por fim, são apresentados os resultados dos testes práticos e as conclusões obtidas.

Capítulo 2
Cinemática de um Robô Móvel

Neste capítulo é definida a representação matemática usada no trabalho e o modelo cinemático para robôs com rodas, como é caso da cadeira de rodas que foi utilizado na implementação do algoritmo.

2.1 Espaço de Configuração e Graus de Liberdade

A configuração de um robô é o menor conjunto de variáveis que especificam, sem ambiguidade, a posição de todos os pontos do robô em um dado instante. O espaço de configuração do robô é o conjunto de todas as possíveis configurações que o robô pode assumir em qualquer instante. Já o número de graus de liberdade (*degrees of freedom*, DoF) do robô é a dimensão do espaço de configuração, ou seja, o menor número de variáveis necessárias para descrever completamente o robô. A escolha de variáveis utilizadas para descrever a configuração pode não ser única, mas a quantidade sempre deverá ser igual ao número de DoF.

A configuração é denotada por um vetor $q \in Q$, em que Q é o espaço de configuração. Todos os demais pontos do robô fazem parte do conjunto $M(q)$, o conjunto de pontos do ambiente ocupados pelo robô que está na configuração q. No exemplo mais simples, um robô com geometria circular com raio r que se movimenta apenas por translação no plano tem como configuração a posição de seu centro $q = [x_c\ y_c]^T$. Todos os pontos (x, y) que satisfazem a expressão $(x - x_c)^2 + (y - y_c)^2 \leq r^2$, pertencem ao robô e, portanto, ao conjunto $M(q)$.

Para robôs articulados, a configuração costuma ser dada pelo deslocamento translacional ou rotacional de cada uma de suas juntas. Para robôs móveis no plano, considerando que sejam tratados como corpos rígidos, a configuração é dada pela posição de um ponto e um ângulo, a orientação.

Como exemplo, consideremos três pontos A, B e C se movendo pelo R^2 de forma independente. Cada um deles têm suas respectivas coordenadas x e y, que podem ser escolhidas livremente. O sistema, portanto, tem 6 DoF. Consideremos agora que esses três pontos não são independentes, eles fazem parte de um mesmo robô planar,

considerado um corpo rígido. O ponto A, (x_A, y_A), pode ser escolhido livremente. Agora, para escolher a posição de B temos que considerar a restrição de rigidez do robô segundo a qual a distância entre os pontos A e B deve ser constante.

$$d(A,B) = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2} \ . \tag{2.1}$$

Essa restrição indica que B deve estar em um círculo de raio $d(A,B)$ com centro em A. Com isso, a única liberdade que temos é escolher o ângulo θ da reta AB em relação a um sistema de coordenadas fixo, sendo:

$$\theta = atan2\left(\frac{y_B - y_A}{x_B - x_A}\right).$$

Para escolher a posição de C temos as restrições de distância constante para A e B:

$$d(A,C) = \sqrt{(x_A - x_C)^2 + (y_A - y_C)^2}, \tag{2.2}$$

$$d(B,C) = \sqrt{(x_B - x_C)^2 + (y_B - y_C)^2}, \tag{2.3}$$

mas com duas restrições, as variáveis x_C e y_C já estão definidas. De fato, com A e B escolhidos todos os demais pontos do robô já estão definidos. Com isso, temos que $q = [x_A, y_A, \theta]^T$ é uma especificação completa para o robô, e dizemos que ele tem 3 DoF.

Com esse exemplo, podemos verificar que os graus de liberdade do robô dependem das suas restrições físicas, mas não necessariamente de todas as restrições. Note que (2.1), (2.2) e (2.3) são exemplos de restrições holonômicas. Essas restrições tem a forma $g(q,t) = 0$, ou seja, dependem apenas das variáveis de configuração q e podem depender do tempo *t*. Cada restrição holonômica linearmente independente elimina um grau de liberdade.

Restrições não-holonômicas, que são restrições de velocidade, tem a forma $g(q, \dot{q}, t) = 0$, dependendo também das derivadas das variáveis de configuração e não é possível integrá-las para obter uma restrição holonômica. Essas restrições não afetam os DoF, mas afetam os graus de mobilidade do robô e o seu conjunto de trajetórias factíveis no espaço de configuração, como visto na seção 2.9 e em [1].

2.2 Planejamento de Trajetória no Espaço de Configuração

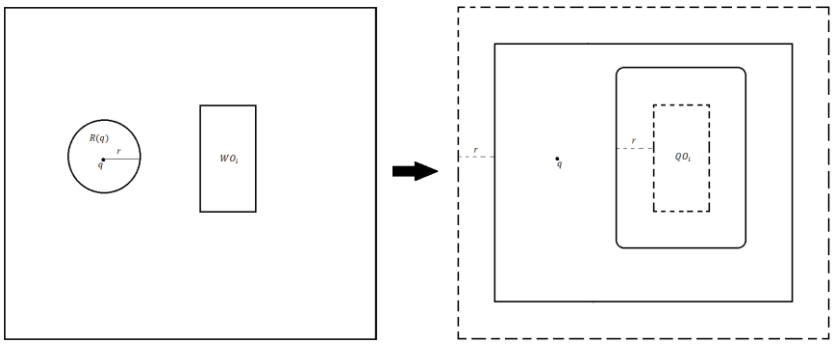
A representação matemática do robô pode ser transferida do espaço operacional, o espaço físico onde o robô de fato trabalha e é tratado como um corpo rígido com área ou volume finito, para o espaço de configuração, em que este é tratado como um ponto. A mesma transferência pode ser feita para os obstáculos que cercam o robô em seu ambiente. Em [1], um obstáculo no espaço de configuração QO_i é definido como um conjunto de configurações do robô para as quais ele intersecta um obstáculo WO_i no espaço operacional:

$$QO_i = \{q \in Q | M(q) \cap WO_i \neq \emptyset\}.$$

O espaço de configuração livre Q_{free} é o conjunto de configurações para os quais o robô não intersecta nenhum obstáculo:

$$Q_{free} = Q \backslash (\cup_i QO_i).$$

A principal dificuldade dos algoritmos de planejamento de trajetória é justamente mapear os obstáculos do espaço operacional para o espaço de configuração para determinar Q_{free}. Mais uma vez, o caso mais simples é o de um robô com geometria circular. Montar Q_{free} para esse robô é simples, pois a distância do centro para todos os pontos na fronteira do robô é constante igual ao raio *r* e com isso basta "expandir" os obstáculos em *r* para criar QO_i, como visto na Figura 2.1.



a) b)

Figura 2.1: a) Espaço Operacional b) Espaço de Configuração.

Para casos ligeiramente mais complexos, como um robô com geometria retangular, a tarefa se torna muito mais difícil, pois a distância entre os pontos da fronteira do robô e os obstáculos depende da orientação do robô.

Com esses conceitos, podemos definir o objetivo do planejamento de trajetória como sendo encontrar um mapeamento continuo $c: [0,1] \rightarrow Q_{free}\ \forall t \in [0,1]$, em que $c[0]$ é a configuração inicial do robô e $c[1]$ é a configuração desejada. A parametrização $[0,1]$ é arbitrária.

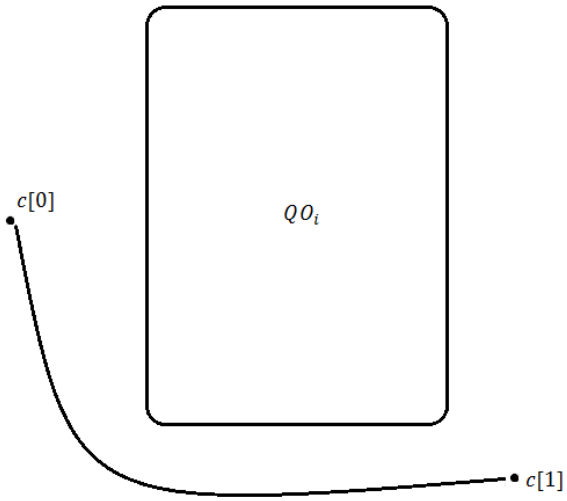


Figura 2.2: Trajetória do robô.

Q_{free} também determina quando uma trajetória é factível. Na Figura 2.3a) o espaço de configuração é conexo (possui apenas um subconjunto) e quando isso ocorre sempre há uma trajetória para o robô. Em 2.3b) para um obstáculo maior, Q_{free} se torna desconexo (possui mais de um subconjunto) e se a configuração inicial e a final não estiverem no mesmo subconjunto, não há trajetória factível.

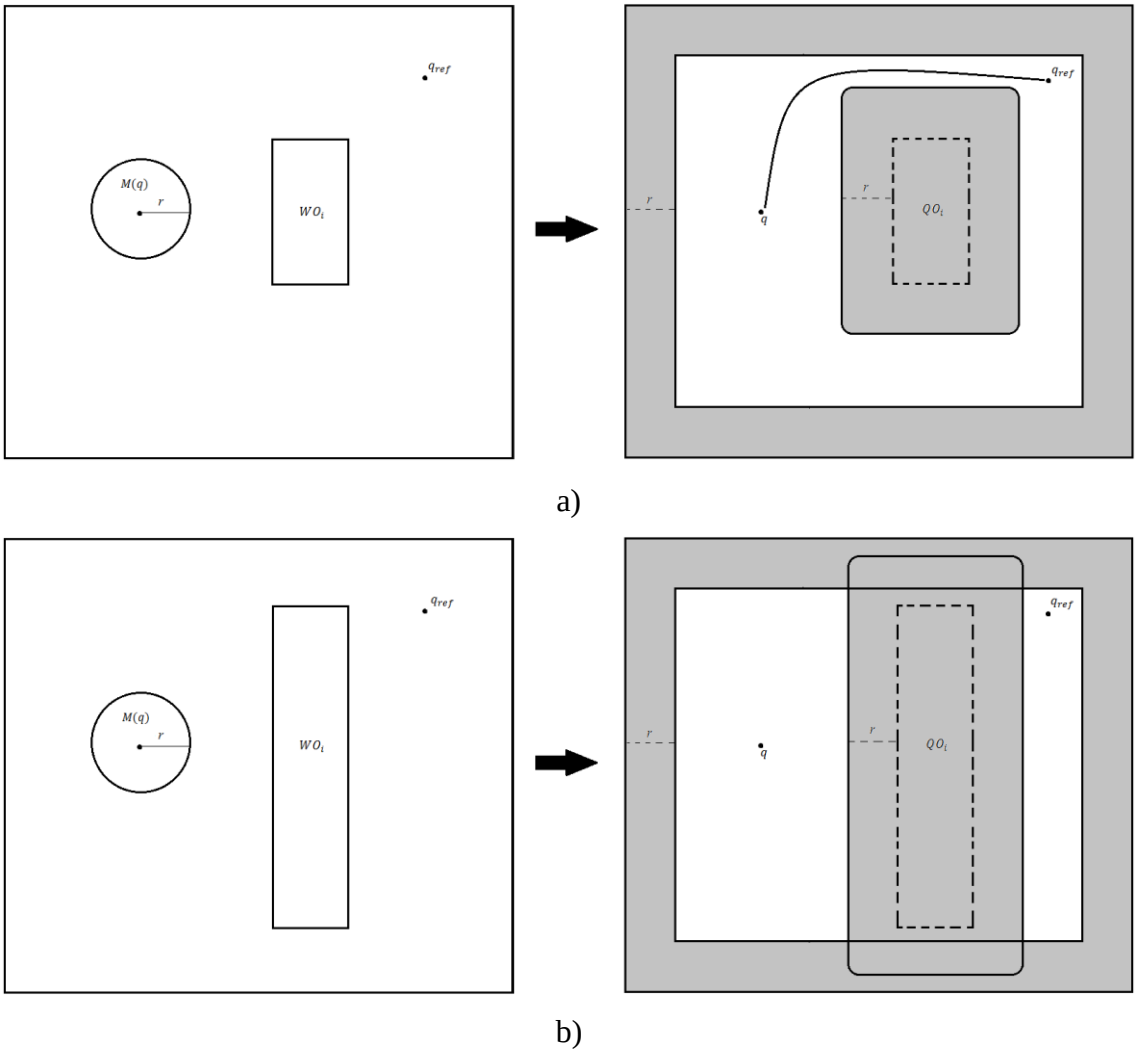


Figura 2.3: a) Espaço de Configuração Conexo b) Espaço de Configuração Desconexo.

2.3 Sistemas de Coordenadas

Na última seção foi encontrada uma configuração para um robô que se move no plano, que consiste nas coordenadas de um ponto e um ângulo. Os valores numéricos dessa configuração, no entanto, dependem do sistema de coordenadas que está sendo usado. Em robótica, é comum que sejam utilizados diversos sistemas de coordenadas. Em robôs articulados, normalmente temos um sistema de coordenadas para cada elo e mais um para a base e outro para o efetuador. Para robôs móveis, há um para o corpo rígido do robô (excluindo as rodas), e um para cada sensor que o robô possua (*laser scanners*, câmeras, *encoders*, IMUs, etc). Além disso, há também pelo menos um sistema de coordenadas inercial.

Quando uma informação de velocidade ou posição é obtida por um sensor, é preciso que o algoritmo de planejamento de trajetória a transfira para o sistema de coordenadas do corpo rígido, que é solidário aos motores para os quais os sinais de controle devem ser enviados.

A mudança de um vetor de um sistema de coordenadas para outro é dada através de uma matriz de rotação.

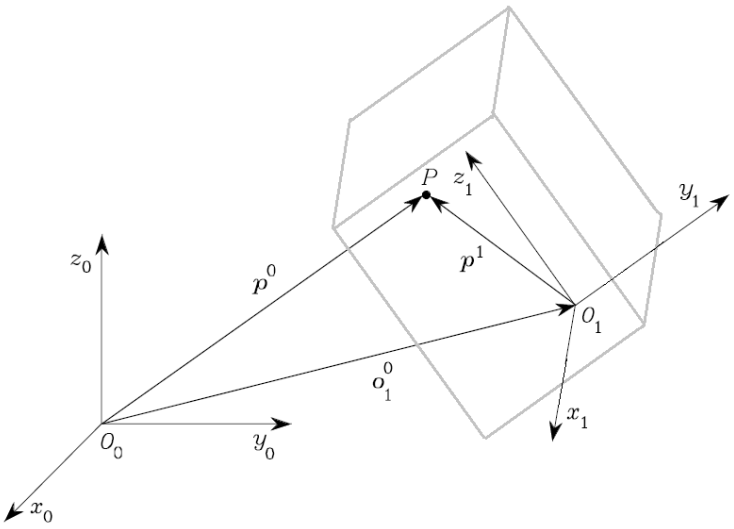


Figura 2.4: Posição e orientação de um corpo rígido.

Na Figura 2.4, temos um corpo rígido no R^3. O sistema de coordenadas $E_0 = \{x_0, y_0, z_0\}$ é inercial e $E_1 = \{x_1, y_1, z_1\}$ é solidário ao corpo.

A translação entre os sistemas é dada pelo vetor o_1^0, que representa a posição de O_1, a origem do sistema solidário, no sistema fixo:

$$o_1^0 = O_{1_x}x_0 + O_{1_y}y_0 + O_{1_z}z_0 \quad \text{ou} \quad o_1^0 = \left[O_{1_x}\, O_{1_y}\, O_{1_z}\right]^T.$$

Já a orientação do sistema solidário em relação ao fixo é dada por:

$$\begin{aligned} x_1 &= (x_1.x_0)x_0 + (x_1.y_0)y_0 + (x_1.z_0)z_0\ , \\ y_1 &= (y_1.x_0)x_0 + (y_1.y_0)y_0 + (y.z_0)z_0\ , \\ z_1 &= (z_1.x_0)x_0 + (z_1.y_0)y_0 + (z_1.z_0)z_0\ . \end{aligned} \tag{2.4}$$

Os produtos escalares são os cossenos diretores dos eixos do sistema solidário em relação aos eixos do sistema fixo.
Então (2.4) pode ser colocada em forma matricial como:

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} = \begin{bmatrix} (x_1.x_0) & (x_1.y_0) & (x_1.z_0) \\ (y_1.x_0) & (y_1.y_0) & (y_1.z_0) \\ (z_1.x_0) & (z_1.y_0) & (z_1.z_0) \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix}.$$

E daí temos que a matriz de rotação R_1^0 é definida como:

$$R_1^0 = [x_1 \quad y_1 \quad z_1] = \begin{bmatrix} (x_1.x_0) & (y_1.x_0) & (z_1.x_0) \\ (x_1.y_0) & (y_1.y_0) & (z_1.y_0) \\ (x_1.z_0) & (y_1.z_0) & (z_1.z_0) \end{bmatrix}, \tag{2.5}$$

onde $$E_0 = R_1^0 E_1 \ . \tag{2.6}$$

A característica especial dessa matriz é que suas colunas são mutuamente ortogonais, já que representam os vetores unitários de um sistema de coordenadas ortonormal. Com isso, surgem 3 restrições devido à ortogonalidade da matriz:

$$\begin{aligned} {x_1}^T y_1 &= 0 \ , \\ {y_1}^T z_1 &= 0 \ , \\ {z_1}^T x_1 &= 0 \ , \end{aligned} \tag{2.7}$$

e mais 3 restrições pois os vetores tem módulo unitário:

$$\begin{aligned} {x_1}^T x_1 &= 1 \ , \\ {y_1}^T y_1 &= 1 \ , \\ {z_1}^T z_1 &= 1 \ . \end{aligned} \tag{2.8}$$

Como consequência de (2.7) e (2.8), $R^T = R^{-1}$ e $\det(R) = 1$ [3].

Em particular temos as rotações elementares, quando uma rotação por um ângulo θ ocorre em apenas um dos eixos fixos.

$$R_z(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

$$R_y(\theta) = \begin{bmatrix} cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{bmatrix}, \tag{2.9}$$

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix}.$$

As rotações elementares são muito úteis já que a partir delas é possivel montar as matrizes para uma rotação em torno de qualquer eixo.

2.3.1 Representações de uma Matriz de Rotação

A matriz de rotação é uma descrição redundante da orientação de um sistema de coordenadas. No R^3, ela possui 9 elementos que não são todos independentes devido as 6 restrições holonômicas independentes em (2.7) e (2.8). Apenas 3 parâmetros, portanto, são necessários para uma representação mínima. De forma similar, no R^2 a matriz possui 4 elementos, mas basta um parâmetro. As duas representações mínimas usadas para o R^3 compostas por três ângulos (chamados ângulos de Euler) são o ZYZ e RPY.

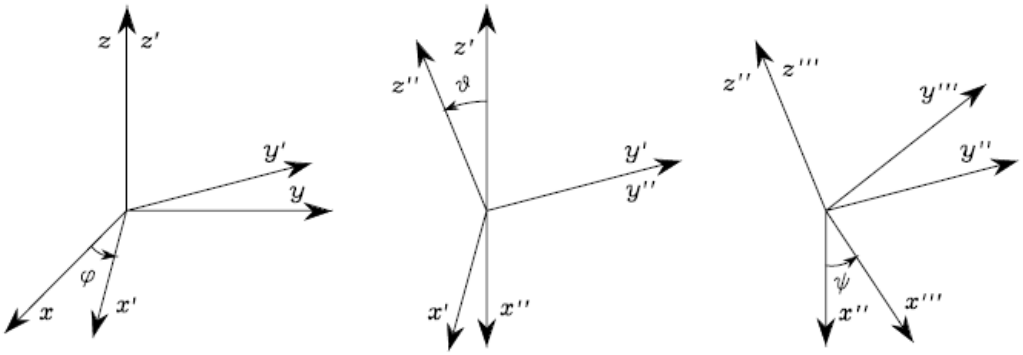


Figura 2.5: Representação ZYZ.

A representação ZYZ consiste dos ângulos $[\varphi, \vartheta, \psi]$ e é mostrada na Figura 2.5. Primeiro ocorre uma rotação por φ em torno de z do sistema de coordenadas inercial, seguida de uma rotação por ϑ em torno do eixo y' do sistema obtido pela rotação φ inicial. Por último, ocorre a rotação por ψ em torno de z'', do sistema obtido pela rotação ϑ.

Como cada uma dessas rotações ocorre em relação ao sistema de coordenadas atual, a matriz R que representa as três transformações é dada pela pós-multiplicação das matrizes de rotação elementares:

$$R(\phi) = R_z(\varphi)\mathrm{R}_{y'}(\vartheta)R_{z''}(\psi) \ ,$$

em que R_z e R_y são as rotações elementares em (2.9) e $R(\phi)$ é:

$$R(\phi) = \begin{bmatrix} c_\varphi c_\vartheta c_\psi - s_\varphi s_\psi & -c_\varphi c_\vartheta s_\psi - s_\varphi c_\psi & c_\varphi s_\vartheta \\ s_\varphi c_\vartheta c_\psi + c_\varphi s_\psi & -s_\varphi c_\vartheta s_\psi + c_\varphi c_\psi & s_\varphi s_\vartheta \\ -s_\vartheta c_\psi & s_\vartheta s_\psi & c_\vartheta \end{bmatrix},$$

em que $c_\varphi = cos(\varphi)$ e $s_\varphi = sin(\varphi)$. O mesmo vale para os ângulos ϑ e ψ.

O problema da representação ZYZ está na singularidade da matriz $R(\phi)$ quando tentamos determinar os valores dos ângulos a partir dela:

Colocando $R(\phi)$ na forma:

$$R(\phi) = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}, \qquad (2.10)$$

temos que:

$$\varphi = atan2(r_{23}, r_{13}) \ ,$$
$$\vartheta = atan2\left(\sqrt{r_{13}^2 + r_{23}^2}, r_{33}\right) \ ,$$
$$\psi = atan2(r_{32}, -r_{31}) \ .$$

Portanto, se $s_\vartheta = 0$ os valores de φ e ψ se tornam indeterminados e apenas a soma ou diferença entre ambos pode ser determinada.

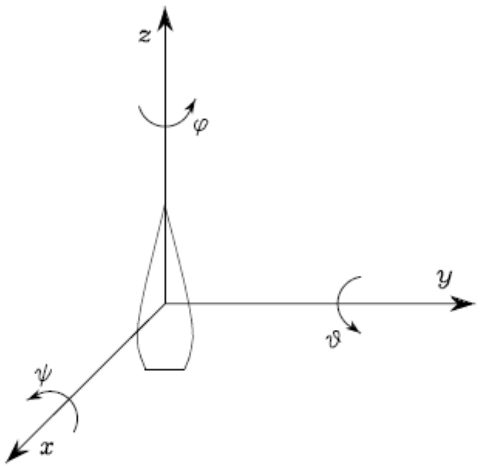


Figura 2.6: Representação RPY

O mesmo problema de singularidade ocorre para a representação mínima RPY (*roll-pitch-yaw*), mostrada na Figura 2.6. Essa representação consiste dos ângulos $[\varphi, \vartheta, \psi]$. Todas as rotações agora ocorrem em relação ao sistema de coordenadas inercial. Primeiro ocorre a rotação por ψ em torno de x, depois a rotação por ϑ em torno de y e por último a rotação por φ em torno de z. Como todas essas rotações ocorrem em relação a um sistema de coordenadas fixo, a matriz R que representa as três rotações é dada pela pré-multiplicação das rotações elementares:

$$R(\phi) = R_z(\varphi)\mathrm{R_y}(\vartheta)R_x(\psi) = \begin{bmatrix} c_\varphi c_\vartheta & c_\varphi s_\vartheta s_\psi - s_\varphi c_\psi & c_\varphi s_\vartheta c_\psi + s_\varphi s_\psi \\ s_\varphi c_\vartheta & s_\varphi s_\vartheta s_\psi + c_\varphi c_\psi & s_\varphi s_\vartheta c_\psi - c_\varphi s_\psi \\ -s_\vartheta & c_\vartheta s_\psi & c_\vartheta c_\psi \end{bmatrix}.$$

Mais uma vez colocando R na forma (2.10), temos:

$$\varphi = atan2(r_{21}, r_{11}) ,$$
$$\vartheta = atan2\left(-r_{31}, \sqrt{r_{32}^2 + r_{33}^2}\right) ,$$
$$\psi = atan2(r_{32}, r_{33}) ,$$

e se $c_\vartheta = 0$, φ e ψ se tornam indeterminados. Apenas a soma ou diferença entre eles pode ser obtida.

Como as representações mínimas possuem problemas de singularidades, uma representação não-mínima que se tornou popular foi o quaternion unitário [3].

O quaternion $q = [q_0\ q_1\ q_2\ q_3]$ é dado por:

$$q_0 = cos\left(\frac{\theta}{2}\right)\ ,\ [q_1\ q_2\ q_3]^T = sin\left(\frac{\theta}{2}\right)w\ ,$$

em que *w* é o eixo de rotação do sistema de coordenadas do robô em relação ao inercial e θ é o deslocamento angular em torno de *w*. Como *q* possui 4 elementos e a representação mínima deve ter 3, o quaternion possui uma restrição de módulo unitário:

$$q_0^2 + q_1^2 + q_2^2 + q_3^2 = 1\ .$$

A matriz de rotação R expressa em função dos elementos do quaternion é:

$$R = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1q_2 - q_0q_3) & 2(q_1q_3 + q_0q_2) \\ 2(q_1q_2 + q_0q_3) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2q_3 - q_0q_1) \\ 2(q_1q_3 - q_0q_2) & 2(q_2q_3 + q_0q_1) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix}\ , \qquad (2.11)$$

e colocando R na forma (2.10), podemos obter o quaternion:

$$\begin{aligned} q_0 &= \tfrac{1}{2}\sqrt{r_{11} + r_{22} + r_{33} + 1}\ , \\ q_1 &= \tfrac{1}{2}sgn(r_{32} - r_{23})\sqrt{r_{11} - r_{22} - r_{33} + 1}\ , \\ q_2 &= \tfrac{1}{2}sgn(r_{13} - r_{31})\sqrt{r_{22} - r_{11} - r_{33} + 1}\ , \\ q_3 &= \tfrac{1}{2}sgn(r_{21} - r_{12})\sqrt{r_{33} - r_{11} - r_{22} + 1}\ . \end{aligned}$$

Podemos verificar que essa representação não tem singularidades.

2.3.2 Representação de um Vetor em mais de um Sistema de Coordenadas

Na Figura 2.4, o ponto P, que pertence ao corpo rígido, tem coordenadas $p^1 = \left[p_x^1\ p_y^1\ p_z^1\right]^T$ no sistema E_1. O objetivo é expressar esse ponto no sistema de coordenadas inercial E_0 no qual as coordenadas são $p^0 = \left[p_x^0\ p_y^0\ p_z^0\right]^T$.

Como p^0 e p^1 são representações do mesmo ponto, podemos escrever:

$$p^0 = (p_x^1 x_1)_0 + (p_y^1 y_1)_0 + (p_z^1 z_1)_0 + O_{1_x} x_0 + O_{1_y} y_0 + O_{1_z} z_0\ ,$$

e por (2.6):

$$p^0 = R_1^0 p^1 + O_1^0\ . \tag{2.12}$$

Quando combinadas, a translação e a rotação de um ponto são chamadas de transformação de coordenadas desse ponto. Uma forma compacta de escrever essa relação é usando a matriz de transformação homogênea

$$T_1^0 = \begin{bmatrix} R_1^0 & O_1^0 \\ 0 & 1 \end{bmatrix}. \tag{2.13}$$

Com isso (2.12) pode ser reescrita como

$$\tilde{p}^0 = T_1^0 \tilde{p}^1\ , \tag{2.14}$$

em que $\tilde{p}^0 = \begin{bmatrix} p^0 \\ 1 \end{bmatrix}$ e $\tilde{p}^1 = \begin{bmatrix} p^1 \\ 1 \end{bmatrix}$

Contudo, a matriz T_1^0 não é ortogonal e, portanto, $T_1^{0^T} \neq T_1^{0^{-1}}$.

2.4 Jacobiano

Assim como é comum em robótica precisar modificar os sistemas de coordenadas nos quais um ponto ou vetor se encontra, também precisamos muitas vezes encontrar a relação entre diferentes representações da configuração de um robô e o modo como a variação das variáveis de uma configuração afeta as da outra, ou seja, é preciso determinar a relação entre as velocidades.

Para o planejamento de trajetória de braços robóticos, o objetivo costuma ser determinar a trajetória do efetuador do braço nas suas coordenadas cartesianas, mas os sinais de controle devem ser enviados aos motores que se encontram em suas juntas e, portanto, é necessário encontrar a relação entre as velocidades das juntas e a velocidade do efetuador, para determinar a lei de controle.

Para robôs móveis que sejam tratados como corpos rígidos, como visto no próximo capítulo, o planejamento de trajetória pode ser feito para alguns pontos específicos do robô, e o objetivo é encontrar as velocidades $(\dot{x}_i, \dot{y}_i)$ de cada um deles. Mas como os sinais de velocidade linear e angular dos motores não agem diretamente nesses pontos específicos, é preciso achar a relação entre $(\dot{x}_i, \dot{y}_i)$ e $\left(\dot{x}, \dot{y}, \dot{\theta}\right)$ do corpo rígido como um todo.

A relação entre duas configurações $x \in X$ e $q \in Q$ é chamada cinemática direta e é dada por:

$$x = k(q) \ . \tag{2.15}$$

Essa relação normalmente é não-linear, e sua derivada é:

$$\dot{x} = \frac{\partial k}{\partial q}\dot{q} = J(q)\dot{q} \ . \tag{2.16}$$

A matriz $J(q)$ é chamada de Jacobiano e é a matriz responsável por mapear as velocidades (e, analogamente, forças, para um modelo dinâmico) do espaço de configuração para o espaço operacional.

2.5 Robôs com Rodas

As rodas são o meio de locomoção mais utilizado para robôs móveis terrestres devido à sua eficiência energética em terrenos planos e implementação simples [4]. Além disso, a questão do equilíbrio do robô não costuma ser um problema no projeto de robôs com rodas, ao contrário do que acontece com robôs que possuem pés, pois no projeto mecânico do robô, as rodas já são desenhadas de forma que toquem a superfície a todo momento, garantindo o seu equilíbrio.

Teoricamente, o equilíbrio estático pode ser garantido com apenas duas rodas, caso o centro de gravidade do robô esteja abaixo do eixo das rodas. Desse modo, a projeção do centro de gravidade na superfície estaria perfeitamente sobre a linha entre os dois pontos das rodas que tocam essa superfície. Na prática, porém, para a maioria dos robôs, seriam necessárias rodas com diâmetro absurdamente grande em relação ao tamanho do robô e ainda assim, devido à inércia, quando o robô saísse do repouso o centro de gravidade sairia dessa linha. Esse equilíbrio, portanto, seria instável. Considera-se, na prática que o menor número de rodas que garantem estabilidade estática ao robô é três, com a condição adicional que o centro de gravidade do robô esteja sobre o triângulo entre seus pontos de contato com a superfície. Para robôs com mais de três rodas, é necessário um sistema de suspensão para garantir que todas toquem a superfície a todo momento [4].

A questão principal para o planejamento de trajetória de robôs com rodas é se a configuração delas (quantidade, tipo e disposição ao longo do robô) permite o controle sobre a velocidade do robô. Cada tipo de roda possui propriedades cinemáticas diferentes que afetam o modelo cinemático do robô como um todo podendo gerar restrições de movimento que determinam as suas trajetórias factíveis.

2.5.1 Cinemática de Robôs com Rodas

Cinemática é o estudo de como sistema mecânicos se comportam. Descreve os movimentos de pontos e corpos rígidos sem considerar as forças que produzem esses movimentos. Esse estudo é suficientemente detalhado para a maioria dos robôs usados em laboratório devido a características como as baixas velocidades, que reduzem os efeitos do deslocamento por inércia quando o sinal de velocidade cessa, e centro de gravidade próximo da superfície, que reduz o risco de capotagem em curvas com ângulo fechado.

Para a modelagem cinemática de um robô com rodas é necessário descrever as restrições geradas por cada uma de suas rodas para o movimento.

Como visto antes, para um robô móvel planar tratado como corpo rígido, há três DoF, e a configuração utilizada é composta pelas coordenadas (x, y) de um ponto do corpo no sistema de coordenadas inercial, que é a origem do sistema de coordenadas solidário ao robô, e a orientação, que é o ângulo entre o sistema de coordenadas do robô e o inercial. Nesse caso, pode-se ignorar os DoF internos do robô como, por exemplo, aqueles devidos aos eixos de rotação das rodas e de suas juntas de direção.

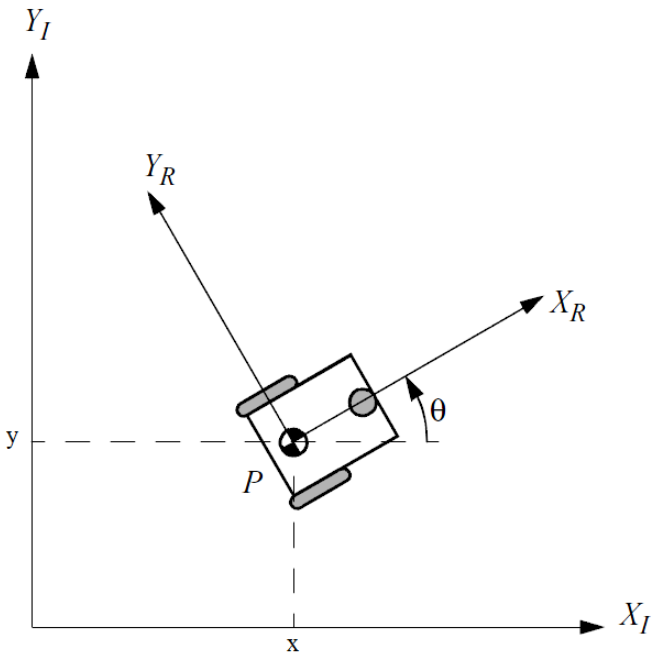


Figura 2.7: Configuração de um robô planar [4].

A notação utilizada para a configuração é $\xi_I = [x\ y\ \theta]^T$.

A matriz de rotação R entre os dois sistemas de coordenadas é dada por uma rotação elementar em z, como em (2.9).

A matriz R também converte a velocidade do robô no seu próprio sistema para o sistema de coordenadas inercial:

$$\dot{\xi}_I = R(\theta)\dot{\xi}_R \tag{2.17}$$

2.5.2 Restrições Cinemáticas das Rodas

Cada tipo de roda tem um conjunto de restrições. Para a determinação dessas restrições são utilizadas duas premissas, [4]:

- O plano perpendicular ao eixo de rotação da roda deve estar sempre na vertical;
- Há apenas um ponto de contato entre a roda e a superfície e não há deslizamento nesse ponto, ou seja, todo deslocamento produzido pela roda é gerado por rolamento ou pela rotação em torno do seu eixo vertical que passa pelo ponto de contato.

Com essas premissas, sempre existem duas restrições para cada roda. Na primeira, todo deslocamento feito na direção em que a roda pode se mover é feito por rolamento puro. Na segunda, como não pode haver deslizamento, a roda não pode sofrer deslocamento na direção perpendicular ao seu plano.

Existem 4 tipos de rodas: padrão, Castor, sueca e esférica. Para esse trabalho são apresentadas apenas a roda padrão e a roda Castor, que são utilizadas na cadeira de rodas usada na implementação do planejamento de trajetória [4].

2.5.2.1 Roda Padrão:

A roda padrão fixa não tem juntas de direção para rotação em torno do seu eixo vertical. Seu ângulo β em relação ao corpo rígido do robô é, portanto, constante e com isso a roda é limitada a se mover para frente e para trás em torno do seu ponto de contato em uma reta com ângulo β no sistema de coordenadas do robô, como mostrado na Figura 2.8. O ponto de contato A da roda é dado em coordenadas polares por l, a distância até a origem e α, seu ângulo em relação ao eixo $+X_R$. O raio da roda é r, o seu deslocamento angular é ϕ e a velocidade linear do ponto de contato é v.

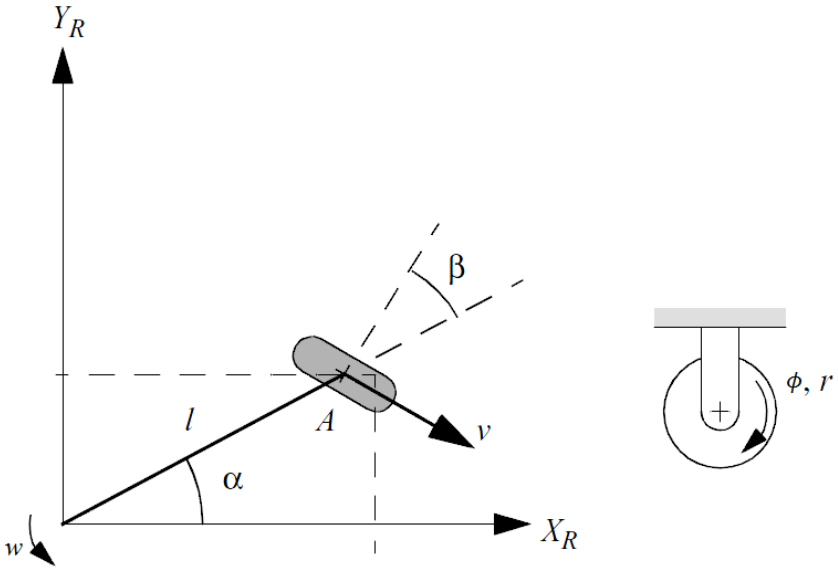


Figura 2.8: Roda padrão [4].

Para a restrição de rolamento puro, todo movimento na direção do plano da roda é dado pelo seu deslocamento angular ϕ.

A relação entre velocidade linear e angular de uma roda é dada por:

$$v = r\dot{\phi} \, .$$

A componente de $\dot{X}_R$ ao longo da reta de v é: $cos\left(\alpha + \beta - \frac{\pi}{2}\right)\dot{X}_R = sin(\alpha + \beta)\dot{X}_R$.

A componente de $\dot{Y}_R$ ao longo da reta de v é: $cos\big(\pi - (\alpha + \beta)\big)\,\dot{Y}_R = -\,cos(\alpha + \beta)\,\dot{Y}_R$.

Existe ainda uma componente da velocidade tangencial sobre a reta de v, gerada pela velocidade angular w do corpo rígido do robô em torno de seu próprio eixo, dada por:

$$l\,cos(\pi - \beta)w = (-l)cos(\beta)w.$$

Somando essas componentes, sendo $\dot{\xi}_R = \left[\dot{X}_R\ \dot{Y}_R\ w\right]^T$, e usando a relação $v = r\dot{\phi}$ obtém-se:

$$[sin(\alpha + \beta)\ -\cos(\alpha + \beta)\ (-l)cos(\beta)]\dot{\xi}_R = r\dot{\phi} \, . \qquad (2.18)$$

Para a restrição de não-deslizamento lateral, a soma de todas as velocidades na direção perpendicular a *v* deve ser nula:

A componente de $\dot{X}_R$ ao longo da reta perpendicular a v é:

$$cos(\alpha + \beta)\dot{X}_R \, .$$

A componente de $\dot{Y}_R$ ao longo da reta perpendicular a *v* é:

$$\sin\big(\pi - (\alpha + \beta)\big)\dot{Y}_R = \sin(\alpha + \beta)\dot{Y}_R \, .$$

E a componente da velocidade tangencial sobre a reta perpendicular a *v* é:

$$l \, sin(\pi - \beta)w = l \, sin(\beta)w \, ,$$

Então, combinando-se as equações anteriores temos:

$$[cos(\alpha + \beta) \ \sin(\alpha + \beta) \ \ l \sin(\beta)]\dot{\xi}_R = 0 \, . \qquad (2.19)$$

Existe também a roda padrão com junta de direção, que permite a rotação da roda em torno do seu eixo vertical, ou seja, $\beta(t)$ passa a ser variável. As restrições para essa roda também são dadas por (2.18) e (2.19).

2.5.2.2 Roda Castor

Essa roda também pode girar em torno do eixo vertical, mas ao contrário da roda padrão com junta de direção, o eixo de rotação vertical não passa pelo ponto de contato entre a roda e a superfície, como mostrado na Figura 2.9.

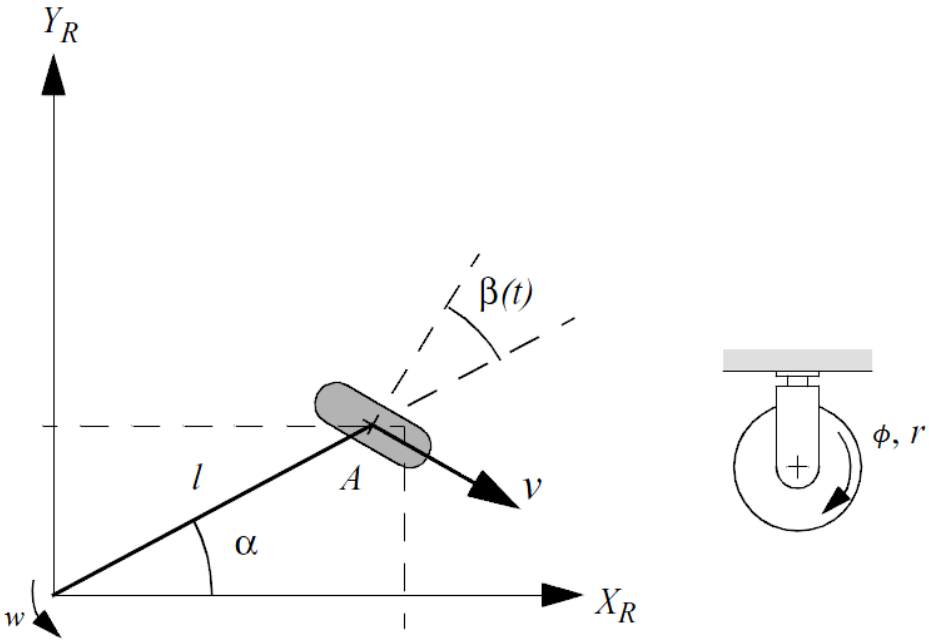


Figura 2.9: Roda Castor [4].

O ponto de contato está em B, que está a uma distância *d* do ponto A que pertence ao eixo de mudança de direção. Os demais parâmetros são definidos da mesma forma que para a roda padrão.

A restrição de rolamento da roda Castor é igual à da roda padrão, pois a única diferença entre as duas, a distância AB, não afeta o movimento na direção do plano da roda. A restrição também é dada por (2.18).

Por outro lado, AB produz uma diferença significativa na restrição de não-deslizamento. As mesmas componentes do movimento na direção perpendicular ao plano que existem na roda padrão existem para a roda Castor, ou seja, $[cos(\alpha+\beta)\ \sin(\alpha+\beta)\ l\sin(\beta)]\dot{\xi}_R$, mas também há uma componente angular dw devido à distância AB, o que produz $[cos(\alpha+\beta)\ \sin(\alpha+\beta)\ (d+l\sin(\beta))]\dot{\xi}_R$. Contudo, essa soma não precisa ser nula, pois o ponto de aplicação dessas componentes é o ponto B, mas o ponto em que a rotação ocorre é A. Isso provoca um movimento dado por $d\dot{\beta}$. Colocando na equação, temos a seguinte restrição de não-deslizamento:

$$[cos(\alpha+\beta)\ \sin(\alpha+\beta)\ (d+l\sin(\beta))]\dot{\xi}_R + d\dot{\beta} = 0. \qquad (2.20)$$

Essa equação indica que qualquer movimento ortogonal ao plano da roda pode ser compensado por um deslocamento de β, e como $\dot{\beta}$ pode assumir qualquer valor,

qualquer movimento lateral é possível. Note que (2.18) e (2.20) indicam que se um robô possui apenas rodas Castor, para qualquer movimento no sistema de coordenadas do robô, existem valores de $\dot{\phi}$ e $\dot{\beta}$ que permitem tal movimento. A roda Castor, portanto, não introduz restrições ao movimento e por isso é considerada uma roda omnidirecional. Outros tipos de rodas omnidirecionais são a roda esférica e a roda sueca [4].

2.5.3 Restrições Cinemáticas do Robô

Conhecendo as restrições de cada roda, podemos agora verificar como o conjunto de rodas afeta a cinemática do corpo rígido. Isso acontece simplesmente combinando todas as restrições existentes. Como visto na seção anterior, apenas as rodas padrão geram restrições. As demais rodas podem ser ignoradas.

Considerando que existem N_f rodas padrão fixas com orientações β_f constantes e deslocamentos angulares $\phi_f(t)$ e N_s rodas padrão com juntas de direção com orientações $\beta_s(t)$ variáveis e deslocamentos angulares $\phi_s(t)$.
As restrições de rolamento de todas as rodas podem ser unidas na expressão:

$$J_1(\beta_s)\dot{\xi}_R = J_2\dot{\varphi}\ , \tag{2.21}$$

onde $J_1(\beta_s) = \begin{bmatrix} J_{1f} \\ J_{1s}(\beta_s) \end{bmatrix}$, J_2 é uma matriz diagonal que contém os raios das rodas, $\dot{\varphi}(t) = \begin{bmatrix} \dot{\phi}_f(t) \\ \dot{\phi}_s(t) \end{bmatrix}$ e J_{1f} e $J_{1s}(\beta_s)$ tem ordem $N_f \times 3$ e $N_s \times 3$, respectivamente, e suas linhas são dadas pela expressão $[sin(\alpha+\beta)\ -\cos(\alpha+\beta)\ (-l)cos(\beta)]$ de (2.18)

Fazemos o mesmo para as restrições de não-deslizamento, obtendo:

$$C(\beta_s)\dot{\xi}_R = 0\ , \tag{2.22}$$

onde $C(\beta_s) = \begin{bmatrix} C_f \\ C_s(\beta_s) \end{bmatrix}$ e C_f e $C_s(\beta_s)$ tem ordem $N_f \times 3$ e $N_s \times 3$, respectivamente, e suas linhas são dadas pela expressão $[cos(\alpha+\beta)\ \sin(\alpha+\beta)\ l\sin(\beta)]$ de (2.19).

Unindo (2.21) e (2.22):

$$\begin{bmatrix} J_1(\beta_s) \\ C(\beta_s) \end{bmatrix} \dot{\xi}_R = \begin{bmatrix} J_2\dot{\varphi} \\ 0 \end{bmatrix}, \tag{2.23}$$

Agora podemos usar (2.23) para modelar a cadeira de rodas. A configuração da cadeira é conhecida como robô diferencial e é mostrada na Figura 2.7. É um robô que possui apenas duas rodas padrão fixas motorizadas com o mesmo eixo de rotação. Como escrito anteriormente, embora seja possível obter equilíbrio estático com apenas duas rodas, o ideal é que existam três ou mais. Para isso, um robô diferencial costuma ter mais algumas rodas Castor não-motorizadas para garantir estabilidade estática (a cadeira de rodas possui quatro). Como elas não são atuadas, não recebem sinais de controle e como não geram restrições, também não aparecem no modelo, sendo, portanto, desprezíveis do ponto de vista do planejamento de trajetória.

Como não há rodas padrão com juntas de direção, $J_1(\beta_s) = J_{1f}$ e $C(\beta_s) = C_f$. Agora precisamos dos parâmetros α, β, l e r de cada roda. Os parâmetros α e β são iguais para todos os robôs diferenciais. Para a roda direita, $\alpha = -\frac{\pi}{2}$ e $\beta = \pi$, e para a roda esquerda, $\alpha = \frac{\pi}{2}$ e $\beta = 0$. Os parâmetros l e r dependem das dimensões do robô usado. Para a cadeira de rodas usada na implementação, $l = 29\ cm$ e *r=17.8 cm* para ambas as rodas. Então:

$$J_1(\beta_s) = \begin{bmatrix} 1 & 0 & l \\ 1 & 0 & -l \end{bmatrix}, \quad C(\beta_s) = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}, \quad J_2 = \begin{bmatrix} r & 0 \\ 0 & r \end{bmatrix},$$

A partir de (2.23) temos:

$$\begin{bmatrix} 1 & 0 & l \\ 1 & 0 & -l \\ 0 & 1 & 0 \end{bmatrix} \dot{\xi}_R = r \begin{bmatrix} \dot{\varphi} \\ 0 \end{bmatrix},$$

$$\dot{\xi}_R = r \begin{bmatrix} 1 & 0 & l \\ 1 & 0 & -l \\ 0 & 1 & 0 \end{bmatrix}^{-1} \begin{bmatrix} \dot{\varphi} \\ 0 \end{bmatrix}$$

$$\dot{\xi}_R = r \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 1 \\ \frac{1}{2l} & -\frac{1}{2l} & 0 \end{bmatrix} \begin{bmatrix} \dot{\varphi} \\ 0 \end{bmatrix}, \tag{2.24}$$

$$\dot{\xi}_R = \frac{r}{2}\begin{bmatrix} \dot{\varphi}_1 + \dot{\varphi}_2 \\ 0 \\ \frac{(\dot{\varphi}_1 - \dot{\varphi}_2)}{l} \end{bmatrix}. \tag{2.25}$$

Usando (2.17) para encontrar a velocidade no sistema inercial, temos:

$$\dot{\xi}_I = \frac{r}{2}\begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} \dot{\varphi}_1 + \dot{\varphi}_2 \\ 0 \\ \frac{(\dot{\varphi}_1 - \dot{\varphi}_2)}{l} \end{bmatrix} = \frac{r}{2}\begin{bmatrix} (\dot{\varphi}_1 + \dot{\varphi}_2)\cos(\theta) \\ (\dot{\varphi}_1 + \dot{\varphi}_2)\sin(\theta) \\ \frac{(\dot{\varphi}_1 - \dot{\varphi}_2)}{l} \end{bmatrix}. \tag{2.26}$$

Verificando (2.24) vemos que a matriz das restrições cinemáticas é um mapeamento entre as velocidades do sistema de coordenadas do robô $\dot{\xi}_R$, que se deseja obter, e as velocidades das rodas $\dot{\varphi}$ sobre as quais realmente podemos atuar com motores. As velocidades das rodas são, portanto, os sinais de controle. Se definirmos:

$$v = \frac{r(\dot{\varphi}_1 + \dot{\varphi}_2)}{2} \quad , \quad w = \frac{r(\dot{\varphi}_1 - \dot{\varphi}_2)}{2l},$$

obtém-se:

$$\dot{\xi}_R = \begin{bmatrix} v \\ 0 \\ w \end{bmatrix} \quad , \quad \dot{\xi}_I = \begin{bmatrix} v\,cos(\theta) \\ v\,sin(\theta) \\ w \end{bmatrix}. \tag{2.27}$$

Um robô capaz de controlar diretamente a velocidade linear para frente *v* e a velocidade angular *w* é o uniciclo, que possui uma única roda padrão com junta de direção. Embora a cadeira de rodas seja um robô diferencial, podemos tratá-la como um uniciclo, pois os seus sinais de entrada são, de fato, *v* e *w*, posteriormente transformados pelo seu *driver* de potência em sinais $\dot{\varphi}_1$ e $\dot{\varphi}_2$ de controle das rodas.

2.6 Mobilidade

A mobilidade de um robô é definida como a sua capacidade de manipular diretamente seus DoF. A única restrição que limita essa capacidade é a de não-deslizamento lateral de todas as rodas, portanto podemos determinar os graus de mobilidade do robô a partir de (2.22). Para que essa equação seja satisfeita, é preciso

que o movimento no sistema de coordenadas do robô esteja sempre no espaço nulo da matriz $C(\beta_s)$. Geometricamente, esse movimento pelo espaço nulo pode ser demonstrado pelo conceito de Centro Instantâneo de Rotação (*Instantaneous Center of Rotation*, ICR). Para cada roda padrão, a restrição de não-deslizamento proíbe o movimento na linha que tem direção perpendicular ao plano da roda em todos os instantes. Como essa restrição vale para todas as rodas, deve haver um único ponto de interseção entre as linhas de não-deslizamento de todas elas, que é o ICR, como mostrado na Figura 2.10, para um veículo de Ackerman.

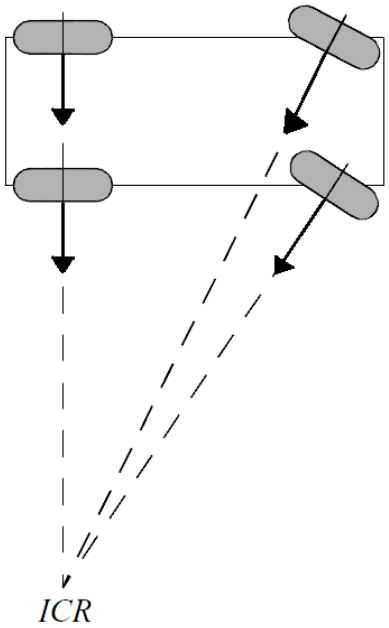


Figura 2.10: Centro Instantâneo de Rotação [4].

Traçando um círculo com centro no ICR para cada roda com seu respectivo raio *r*, vemos que a velocidade linear da roda é tangente ao círculo, não tendo componente centrípeta, o que comprova que essa é a solução geométrica para (2.22). Caso todas as rodas estejam se movendo na mesma direção, consideramos que $r \to \infty$.

Com o ICR, podemos ver que a mobilidade do robô não depende do número de rodas e sim do número de restrições de não-deslizamento (não-holonômicas) independentes. Desse modo, são necessárias apenas duas linhas em direções diferentes (linhas independentes) para determinar um ponto de interseção. Na Figura 2.10, as duas rodas traseiras tem a mesma linha de não-deslizamento, e somente uma delas é independente. Escolhendo a linha de uma das rodas da frente, obtemos o ICR e com isso a linha da quarta roda já está determinada e, portanto, ela também não é independente.

Para o robô diferencial, como a cadeira de rodas, as duas rodas padrão fixas estão alinhadas e apenas uma delas é independente e o ICR está no infinito.

Além da análise geométrica por ICR também é possível verificar matematicamente os graus de mobilidade pelo número de linhas independentes da matriz $C(\beta_s)$, ou seja, seu posto $rank[C(\beta_s)]$.

A matriz $C(\beta_s)$ tem ordem $(N_f + N_s) \times 3$, onde 3 é o número de DOF do robô no plano. Com isso, $0 \leq rank[C(\beta_s)] \leq 3$. Se $rank[C(\beta_s)] = 0$, não há restrições de não-deslizamento e o robô pode ter somente rodas omnidirecionais. Se $rank[C(\beta_s)] = 3$, o robô não pode se mover em nenhum dos DoF. Com isso, o grau de mobilidade do robô δ_m é definido como:

$$\delta_m = dim\ N[C(\beta_s)] = 3 - rank[C(\beta_s)].$$

Portanto, para o veículo de Ackerman da Figura 2.10, $\delta_m = 1$ e para o robô diferencial $\delta_m = 2$. Os graus manipulados diretamente pelo robô diferencial através das velocidades angulares das rodas são o deslocamento em X_R e a orientação. Por outro lado, ele não é capaz de manipular Y_R.

2.7 Dirigibilidade

Além da mobilidade, um robô também pode ser capaz de manipular indiretamente alguns DoF utilizando primeiro as juntas de direção de suas rodas. A essa capacidade dá-se o nome de dirigibilidade. A manipulação é indireta porque, após a manipulação desse DoF é preciso manipular algum dos DoF da mobilidade para que a mudança de direção das rodas gere impacto no movimento do corpo rígido do robô.
O grau de dirigibilidade de um robô δ_s é definido como:

$$\delta_s = rank[C_s(\beta_s)].$$

Como $C_s(\beta_s)$ é uma parcela de $C(\beta_s)$, uma roda padrão com junta de direção pode reduzir a mobilidade e aumentar a dirigibilidade do robô. Sua orientação atual produz uma restrição de não-deslizamento, mas a capacidade de mudar essa orientação permite trajetórias adicionais no futuro. No caso do veículo de Ackerman, escolhendo a

direção de uma das rodas da frente, a direção da outra já está determinada e, portanto, $\delta_s = 1$. Para o robô diferencial, não existem rodas com juntas e $\delta_s = 0$.

2.8 Manobrabilidade

A manobrabilidade é o conjunto completo de DoF que o robô pode manipular, ou seja, o grau de manobrabilidade δ_M é definido como:

$$\delta_M = \delta_m + \delta_s.$$

Inclui, portanto, todos os DoF manipulados diretamente pela velocidade angular das rodas e os DoF manipulados indiretamente pelas juntas de direção das rodas. Para o robô diferencial, $\delta_M = 2$.

2.9 Trajetórias Factíveis

A manobrabilidade representa todos os DoF que podem ser manipulados pelo robô e, portanto são os possíveis sinais de saida da lei de controle responsável por fazer o robô seguir uma determinada trajetória. Então, é necessário conhecer os graus de manobrabilidade para determinar o conjunto de trajetórias que podem ser planejadas para o robô.

O número de velocidades independentes que podem ser controladas pelo robô é dado pelo seu grau de mobilidade δ_m. A questão é saber como δ_m afeta a trajetória do robô. Para um robô omnidirecional, $\delta_m = 3$, e com isso ele pode atingir qualquer configuração final em Q_{free} (considerando que este seja conexo) atuando sobre os três DoF simultaneamente, logo, todas as trajetórias que ligam a configuração inicial à final são factíveis. Já para robôs não-omnidirecionais, isso não ocorre. O exemplo clássico é um carro (que tem a configuração do veículo de Ackerman) que deve estacionar em uma vaga que está ao seu lado. Como não pode andar lateralmente, como um robô omnidirecional, não pode simplesmente andar em trajetória reta até a vaga. Ainda assim, o carro com $\delta_m = 1$ é capaz de atingir essa configuração por inúmeras outras trajetórias, como dando a ré inicialmente e então lentamente alinhar o carro para frente e com mudança de direção das juntas das rodas. O robô diferencial, com $\delta_m = 2$, também

não pode simplesmente se mover lateralmente em linha reta, mas pode estacionar mudando sua orientação até estar de frente para a vaga e então andar para frente e voltar para a orientação inicial. Para os três exemplos mencionados, $DoF = 3$ e com isso qualquer configuração final pode ser obtida a partir de qualquer configuração inicial. A diferença é que δ_m determina o conjunto de trajetórias factíveis que ligam essas duas configurações.

Capítulo 3
Planejamento de Trajetória

Como visto no Capítulo 2, o objetivo básico de um algoritmo para planejamento de trajetória é encontrar um caminho contínuo entre a configuração inicial de um robô e a configuração final desejada no espaço de configuração desse robô. Os algoritmos de navegação normalmente são divididos em dois grupos:

- Métodos *off-line*, que primeiro precisam determinar todo o espaço de configuração do robô, para então calcular a trajetória. Não há integração entre o planejamento e a execução da trajetória. Nesses métodos, a trajetória que o robô deve seguir é totalmente planejada antes de sua execução começar, com base nas informações que o robô obtém sobre o ambiente até esse momento. Informações novas obtidas pelos sensores durante a execução são ignoradas. A desvantagem desses métodos é que funcionam apenas se todos os obstáculos do ambiente forem estáticos ou com leis de movimento conhecidas. A execução, então, consiste de uma malha fechada de controle que minimize o erro entre a configuração atual do robô e a referência daquele instante ao longo da trajetória. O tempo necessário para o planejamento *off-line* de trajetória é várias ordens de grandeza mais lento que o tempo de resposta típico de um robô, tornando sua interação com o ambiente mais lenta, e tornando mais limitado seu uso para sistemas em tempo-real, que exijam alta precisão [5]. Outro problema desses métodos é que determinar o espaço de configuração a partir do espaço operacional é um problema complexo. Mesmo para robôs que se deslocam no plano, apenas para um robô de geometria circular tendo as coordenadas do seu centro como configuração é simples montar o espaço de configuração, como visto na seção 2.2.

- Métodos *online*, capazes de desviar de obstáculos no espaço de configuração mesmo sem conhecer suas formas *a priori*. A maioria desses métodos, porém, funciona apenas para algumas classes específicas de robôs, como os métodos *Bug*, explicados em [1], que funcionam apenas para robôs planares.

Para esse trabalho, utilizamos o método dos Campos Potenciais Artificiais, chamado de APF a partir de agora (do inglês, *Artificial Potential Fields*), que é um método *online* de controle reativo (o sinal de controle depende de mudanças no ambiente) muito popular na comunidade acadêmica e que pode ser usado para uma grande variedade de robôs (móveis ou braços robóticos, planares ou espaciais). Uma característica desse método é que o planejamento e a execução da trajetória ocorrem simultaneamente, pois ao invés de montar o espaço de configuração a partir do ambiente (que não precisa ser conhecido *a priori*), esse método utiliza os sensores do robô para explorar apenas os obstáculos do ambiente próximos ao robô e gerar os sinais de controle a cada instante, aproximando o robô da referência enquanto evita os obstáculos. Dessa forma, a trajetória é construída incrementalmente e não existe uma malha de controle para rastreamento de referência como ocorre nos métodos *off-line*. O APF age simultaneamente como planejador de trajetória e lei de controle do robô [4].

3.1 O Método APF

Inicialmente tratamos o robô como um ponto q no espaço de configuração e depois expandiremos a teoria para o caso em que o robô seja tratado como um corpo rígido no seu espaço operacional.

O APF trata o robô como uma partícula que se move em um campo potencial. O campo potencial é descrito matematicamente por uma função potencial, que é uma função real e diferenciável $U: R_m \rightarrow R$. Fisicamente, costuma-se fazer uma analogia entre função potencial e a energia associada a uma partícula ou corpo. O robô é uma partícula com carga positiva e a referência é uma partícula com carga negativa que, consequentemente, atrai o robô para si devido ao seu campo potencial atrativo. Os obstáculos, por outro lado, são vistos como cargas positivas que, por terem o mesmo sinal que o robô, exercem um campo potencial repulsivo sobre ele, afastando-o. O processo é ilustrado na Figura 3.1.

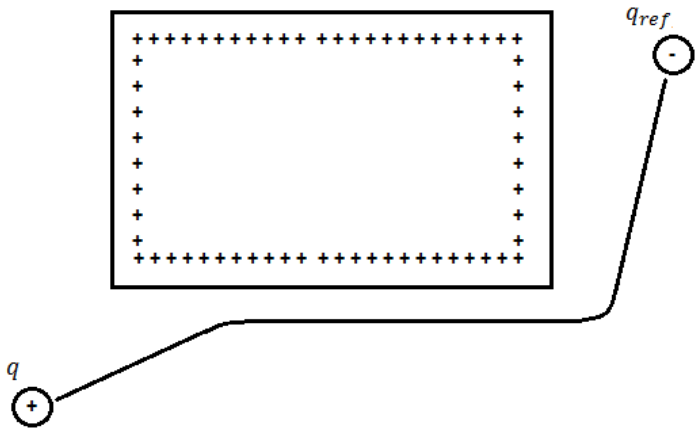


Figura 3.1: Trajetória obtida por um campo atrativo da referência e um campo repulsivo do obstáculo.

É esperado que a soma dos potenciais repulsivo e atrativo levem o robô até a referência em uma trajetória decrescente de potencial que impeça o robô de atingir qualquer obstáculo. Essa técnica é conhecida como descida do gradiente (*Gradient Descent*) e a lei de controle é dada por:

$$v = -\nabla U(q). \tag{3.1}$$

A função gradiente é a generalização da função derivada para funções multivariáveis e é representada pela notação:

$$\nabla U(q) = \left[\frac{\partial U}{\partial q_1}(q), \dots, \frac{\partial U}{\partial q_m}(q)\right]^T.$$

Usando a analogia entre função potencial e energia, o gradiente da função potencial $\nabla U(q)$ costuma ser visto como uma força, que é o gradiente da energia. Em cada ponto, $\nabla U(q)$ aponta para a direção que maximiza localmente o valor de $U(q)$, partindo desse ponto.

O gradiente de uma função, assim como a derivada, é uma função linear, e satisfaz o princípio da superposição:

$$\nabla\big(f(q) + g(q)\big) = \nabla f(q) + \nabla g(q).$$

Como os sinais de controle da cadeira de rodas usada na implementação são velocidades, utilizamos o modelo cinemático para o robô, e o gradiente é tratado como um vetor de velocidade *v* e não de força.

Como o gradiente aponta para a direção na qual o potencial mais cresce, $-\nabla U(q)$ aponta para a direção na qual o potencial mais diminui. Com a lei de controle (3.1), o robô segue pela descida do gradiente e irá parar apenas ao atingir um valor de q no qual o gradiente seja nulo, ou seja, um ponto crítico da função potencial. A função potencial atrativa da referência deve ser escolhida de tal forma que a referência seja um mínimo local dessa função. Embora matematicamente um ponto crítico não precise ser necessariamente um mínimo local da função potencial (também pode ser um máximo local ou um ponto de sela), como a lei de controle força o robô a sempre buscar o potencial mais baixo, a única forma dele estar em ponto de sela ou em um máximo é se essa for a configuração inicial, o que é improvável. Mesmo que isso aconteça, esses são pontos de equilíbrio instável e qualquer perturbação induzirá movimento ao robô. O ponto de mínimo, por outro lado, é estável, e uma vez que o robô o atinja, a lei de controle não permite que ele se afaste.

3.1.1 Escolha das Funções Potenciais

A função potencial $U(q)$ a qual o robô está submetido possui duas componentes:

$$U(q) = U_{att}(q) + U_{rep}(q), \tag{3.2}$$

na qual $U_{att}(q)$ é o potencial atrativo da referência e $U_{rep}(q)$ é a soma dos potenciais repulsivos dos obstáculos.

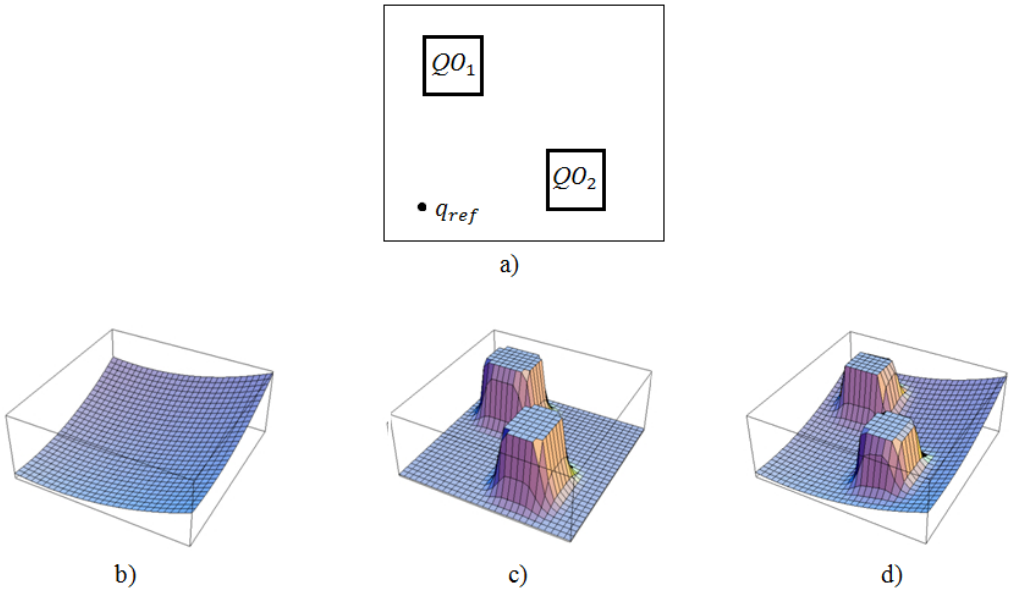


Figura 3.2: a) O espaço de configuração b) Campo atrativo da referência c) Campos repulsivos dos obstáculos d) Soma dos Campos.

Não existe uma única escolha para as funções $U_{att}(q)$ e $U_{rep}(q)$. Qualquer função que satisfaça as condições definidas a seguir pode ser utilizada. Além disso, pela definição de função potencial, $U_{att}(q)$ e $U_{rep}(q)$ devem ser diferenciáveis para todo $q \in Q_{free}$.

Para $U_{att}(q)$, é necessário que a função seja monotonicamente crescente em relação à distância entre o robô e a referência, ou seja, quanto maior a distância Euclidiana entre q e q_{ref}, maior o valor de $U_{att}(q)$. Também é necessário que $U_{att}(q)$ seja definida positiva, ou seja, $U_{att}(q) > 0$, para todo $q \neq q_{ref}$ e $U_{att}(q_{ref}) = 0$. Essas condições garantem que a função tem um único mínimo global em $q = q_{ref}$.
Dadas as condições, a função mais simples a ser usada é o potencial cônico [1]

$$U_{att}(q) = \xi\, d(q, q_{ref}), \tag{3.3}$$

sendo $d(q, q_{ref})$ a distância Euclidiana entre q e q_{ref} e ξ uma constante real usada para escolher a escala do gradiente atrativo:

$$\nabla U_{att}(q) = \frac{\xi}{d(q,q_{ref})}(q - q_{ref}), \tag{3.4}$$

que tem módulo igual a ξ. Contudo, esse gradiente não está definido justamente para $q = q_{ref}$. Essa descontinuidade causa um problema de *chattering* quando o robô se aproxima da referência. Por esse motivo seria melhor encontrar uma função que seja continuamente diferenciável, embora essa característica não seja obrigatória para uma função potencial atrativa.

A segunda opção é uma função quadrática:

$$U_{att}(q) = \frac{\xi}{2} d^2(q, q_{ref}), \tag{3.5}$$

com gradiente dado por:

$$\nabla U_{att}(q) = \xi(q - q_{ref}). \tag{3.6}$$

Esse gradiente diminui linearmente com a distância até q_{ref}, o que é uma característica desejável, já que o robô se aproxima mais rapidamente da referência no começo pois o gradiente é maior, e se aproxima mais devagar quando está mais perto, o que reduz o sobrepasso provocado pela quantização de passo do computador. Além disso, esse gradiente não possui descontinuidades. O único problema desse gradiente é que, por não ser limitado, o sinal de controle pode saturar. Isso pode ser resolvido fazendo um chaveamento entre o potencial quadrático e o potencial cônico [1]:

$$U_{att}(q) = \begin{cases} \frac{\xi}{2} d^2(q, q_{ref}), & d(q, q_{ref}) \leq d^*_{ref}, \\ d^*_{ref}\, \xi\, d(q, q_{ref}) - \frac{\xi}{2}(d^*_{ref})^2, & d(q, q_{ref}) > d^*_{ref}, \end{cases} \tag{3.7}$$

$$\nabla U_{att}(q) = \begin{cases} \xi(q - q_{ref}), & d(q, q_{ref}) \leq d^*_{ref}, \\ \frac{d^*_{ref}\, \xi\, (q - q_{ref})}{d(q, q_{ref})}, & d(q, q_{ref}) > d^*_{ref}, \end{cases} \tag{3.8}$$

onde d^*_{ref} é o limite em que ocorre o chaveamento. O termo adicional no potencial cônico em (3.7) é apenas uma constante para que a continuidade seja mantida, assim como o d^*_{ref} que multiplica $\xi d(q, q_{ref})$ é para manter a continuidade do gradiente. A equação (3.7) foi a função escolhida para esse trabalho.

Para $U_{rep_i}(q)$, a função deve ser monotonicamente crescente em relação ao inverso da distância entre o robô e o obstáculo *i*, ou seja quanto mais próximo do obstáculo, maior o potencial. Além disso, ao contrário do potencial atrativo, não é necessário que o potencial repulsivo tenha alcance infinito, ele pode afetar a trajetória do robô apenas se estiver a uma dada distância do obstáculo. Para distâncias maiores que esse limite estabelecido, $U_{rep_i}(q) = 0$, e para distâncias menores que o limite $U_{rep_i}(q) > 0$. Com essas condições podemos utilizar a função:

$$U_{rep_i}(q) = \begin{cases} \frac{\eta}{2}\left(\frac{1}{d_i(q)} - \frac{1}{Q_i^{max}}\right)^2, & d_i(q) \leq Q_i^{max}, \\ 0, & d_i(q) > Q_i^{max}, \end{cases} \tag{3.9}$$

$$\nabla U_{rep_i}(q) = \begin{cases} \eta\left(\frac{1}{Q_i^{max}} - \frac{1}{d_i(q)}\right)\frac{\nabla d_i(q)}{d_i(q)^2}, & d_i(q) \leq Q_i^{max}, \\ 0, & d_i(q) > Q_i^{max}, \end{cases} \tag{3.10}$$

$$d_i(q) = \min_{c \in QO_i} d(q - c) \qquad \text{e} \qquad \nabla d_i(q) = \frac{q-c}{d(q,c)}$$

onde η é uma constante para a escala do gradiente repulsivo, Q_i^{max} é o limite da influência do potencial repulsivo e $d_i(q)$ é a distância mínima do robô até o obstáculo *i*, como ilustrado na Figura 3.3, em que c_0 é o ponto do obstáculo que está localizado a essa distância mínima.

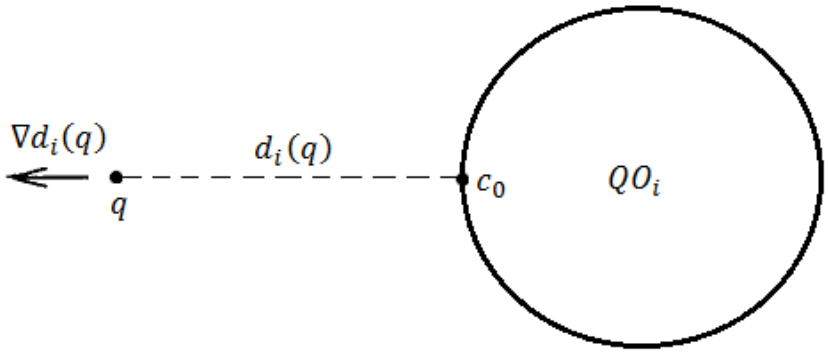


Figura 3.3: Distância mínima até um obstáculo.

Essa função potencial garante que o sinal de controle afaste fortemente o robô do obstáculo para distâncias curtas e de forma suave para distâncias maiores.
O potencial de todos os obstáculos é:

$$U_{rep}(q) = \sum_{i=1}^{n} U_{rep_i}(q) \tag{3.11}$$

Quanto à implementação do algoritmo, o potencial atrativo pode ser facilmente calculado se o robô conhece (i) a referência, o que normalmente ocorre já que a referência é a entrada do programa; (ii) a sua própria configuração, o que é um pouco mais complexo e exige um algoritmo de localização para o robô. O algoritmo de localização adotado nesse trabalho é apresentado no Capítulo 4.

Já para o potencial repulsivo, o problema é determinar $d_i(q)$, o vetor que aponta para o ponto mais próximo do obstáculo. Caso o robô tenha um *laser scanner*, esse vetor pode ser facilmente obtido já que os pontos do *laser* já são dados nas coordenadas polares e, portanto, basta verificar quais pontos tem o valor do raio menor que seus vizinhos. A implementação é discutida no Capítulo 6.

3.1.2 Problema dos Mínimos Locais

O APF é um algoritmo de simples implementação, exige pouco esforço computacional, funciona tanto para robôs que se deslocam no plano quanto no espaço e que tenham as formas mais diversas e também funciona para obstáculos estáticos e dinâmicos, já que os potenciais variam no tempo. Apesar dessas vantagens, existe uma falha inerente ao método.

Vimos que a lei de controle do APF sempre encaminha o robô para um mínimo da função potencial e esta é projetada de forma que a referência sempre seja um ponto de potencial mínimo, mas não é possível garantir que ela seja o único mínimo. Isso significa que dependendo da configuração inicial do robô e da disposição dos obstáculos no ambiente, ele pode não atingir a referência, mesmo que exista uma trajetória factível [6], como visto na Figura 3.4.

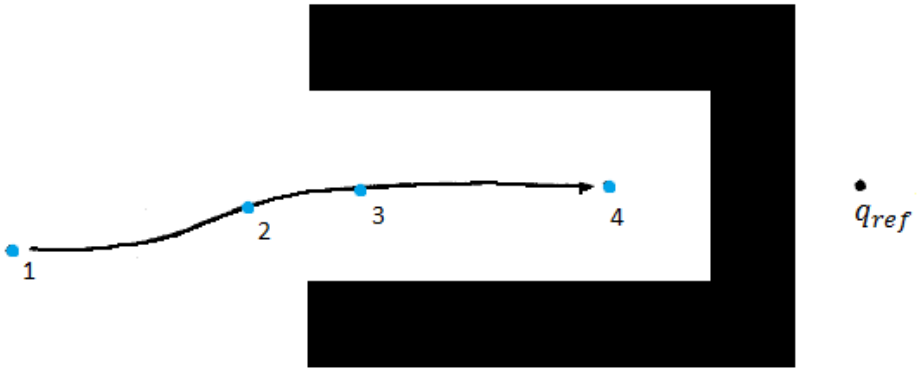
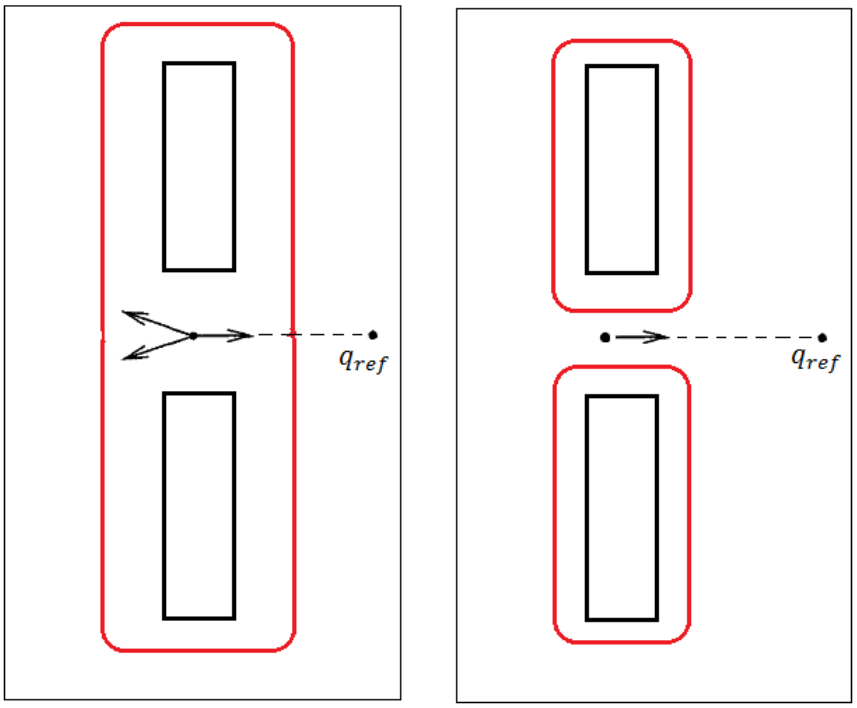


Figura 3.4: Mínimo local devido a um obstáculo côncavo [1].

Inicialmente o robô é atraído apenas pelo campo atrativo da referência (ponto 1) e se aproxima desta, até que o robô entre na zona de influência da parte inferior do obstáculo (ponto 2), que repele o robô para cima até que ele seja influenciado igualmente pela repulsão das partes inferior e superior do obstáculo (ponto 3), deslocando-se horizontalmente até a referência. Em um momento, o robô começa a ser influenciado também pelo campo repulsivo da parte central do obstáculo que gera um sinal de controle na mesma direção, mas com sentido contrário ao de atração. Em certo momento, o gradiente do campo atrativo, que diminui com a aproximação, passa a ter o mesmo módulo que o repulsivo, que aumenta com a aproximação, até que o gradiente resultante seja nulo em um mínimo local que não é a referência (ponto 4).

Por outro lado, se o robô tivesse começado em um ponto acima do obstáculo, ele conseguiria atingir a referência. O problema dos mínimos locais não ocorre apenas para obstáculos côncavos como na Figura 3.4. A Figura 3.5 a) mostra uma situação na qual o robô tenta passar entre dois obstáculos e pode não atingir a referência, pois as componentes horizontais somadas dos dois sinais repulsivos podem ter módulo igual ao do sinal atrativo.



a) b)

Figura 3.5: a) Mínimo local devido à passagem entre dois obstáculos b) Passagem livre.

Para esse caso, o problema poderia ser resolvido alterando os parâmetros η, ξ e Q_i^{max} das funções potencias (3.7) e (3.9). Ainda na Figura 3.5 a) se ξ for aumentado e η for diminuído, a soma das componentes horizontais não será suficiente para parar o robô. Na figura 3.5 b) o problema é resolvido reduzindo a zona de influência Q_i^{max} dos campos repulsivos, embora com isso o risco de o robô atingir o obstáculo aumente caso a sua inércia não seja desprezível.

Existem formas de resolver o problema dos mínimos locais, como as funções de navegação e o algoritmo *brushfire*, [1][6], mas esses métodos exigem que o espaço de configuração seja conhecido, para calcular a descida do gradiente até a referência antes que o robô comece a se movimentar. Esses métodos são variantes *off-line* do APF.

3.2 O APF para corpos rígidos

Até agora, tratamos o robô como um ponto no espaço de configuração. Como dito anteriormente, é difícil montar esse espaço para a maioria dos robôs, principalmente quando ele é não-Euclidiano (não é do tipo R^n) e multidimensional. A partir de agora, utilizaremos o APF para gerar os campos potenciais no espaço operacional do robô (que é sempre Euclidiano), tratando-o como um corpo rígido e depois transferir os gradientes encontrados para o espaço de configuração. Agora devemos pensar nos gradientes como forças e não velocidades e estabeleceremos a relação entre as forças do espaço operacional e do espaço de configuração através do Jacobiano.

Seja x a representação de um ponto qualquer do corpo rígido no espaço operacional e q a configuração do robô. No Capítulo 2 vimos que existe a relação de cinemática direta $x = k(q)$. f e u são as forças generalizadas (forças e torques) geradas pelos campos potenciais sobre o robô no espaço operacional e no espaço de configuração, respectivamente. O objetivo é encontrar uma relação entre u e f. Para isso usamos o princípio dos trabalhos virtuais, segundo o qual a energia ou potência, sendo grandezas escalares, são independentes do sistema de coordenadas utilizado. Isso significa que a potência gerada pela força f em x deve ser igual a potência gerada pela força u em q. A potência no espaço operacional é dada pelo produto entre força e velocidade $f^T\dot{x}$ e, no espaço de configuração, por $u^T\dot{q}$.

$$f^T\dot{x} = u^T\dot{q}$$

E como também visto no Capítulo 2, a diferenciação da relação de cinemática direta leva à relação $\dot{x} = J\dot{q}$ onde J é o Jacobiano. Com isso obtém-se:

$$f^T J\dot{q} = u^T\dot{q} \rightarrow f^T J = u^T$$

$$u = J^T f \tag{3.12}$$

A cadeira de rodas usada na implementação é tratada como um retângulo se movendo no plano. Usando as notações da Figura 3.6 podemos calcular as forças generalizadas no espaço de configuração caso uma força f seja exercida em um dos vértices da cadeira.

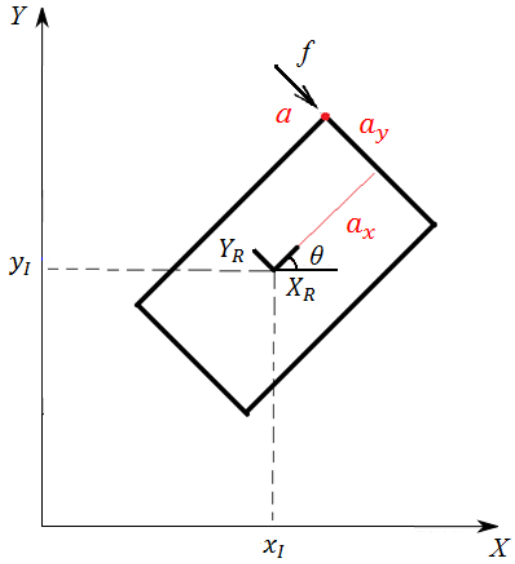


Figura 3.6: Força sobre um vértice do robô.

O vértice a tem coordenadas (a_x, a_y) no sistema de coordenas da cadeira e a origem desse sistema de coordenadas está em (x_i, y_i) no sistema de coordenadas inercial. A orientação do sistema de coordenadas da cadeira em relação ao sistema de coordenadas inercial é denotada pelo ângulo θ.

A cinemática direta $x_a = k(q)$ é dada por:

$$k(q) = \begin{bmatrix} x_i + a_x \cos(\theta) - a_y \sin(\theta) \\ y_i + a_x \sin(\theta) + a_y \cos(\theta) \end{bmatrix}, \tag{3.13}$$

e o Jacobiano por:

$$J(q) = \frac{\partial k}{\partial q}(q) = \begin{bmatrix} 1 & 0 & -a_x \sin(\theta) - a_y \cos(\theta) \\ 0 & 1 & a_x \cos(\theta) - a_y \sin(\theta) \end{bmatrix}. \tag{3.14}$$

Substituindo (3.13) e (3.14) em (3.12), a força *u* no espaço de configuração é:

$$\begin{bmatrix} u_x \\ u_y \\ u_\theta \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ -a_x \sin(\theta) - a_y \cos(\theta) & a_x \cos(\theta) - a_y \sin(\theta) \end{bmatrix} \begin{bmatrix} f_x \\ f_y \end{bmatrix},$$

ou pode ser expressa como:

$$\begin{bmatrix} u_x \\ u_y \\ u_\theta \end{bmatrix} = \begin{bmatrix} f_x \\ f_y \\ -f_x\big(a_x \sin(\theta) + a_y \cos(\theta)\big) + f_y\big(a_x \cos(\theta) - a_y \sin(\theta)\big) \end{bmatrix}. \quad (3.15)$$

Essa expressão indica que a força *f* no vértice *a* gera uma força similar no eixo do robô. A diferença é que também surge um torque dado por u_θ devido à distância entre os pontos de aplicação de *f* e *u*.

Resta agora determinar a força *f* gerada pelos campos potenciais. Para o espaço de configuração era simples determinar as forças (que são os gradientes), pois um único ponto sofria a ação dos campos potenciais. Para um corpo rígido no plano existem infinitos pontos, portanto o cálculo do efeito do campo sobre o corpo exigiria a computação da integral do efeito do campo sobre cada ponto, o que é uma solução matematicamente e computacionalmente complexa.

Uma solução mais simples é discretizar a solução anterior, selecionando apenas um pequeno conjunto de pontos significativos do robô e calcular o gradiente do campo sobre cada um deles. O cálculo dos gradientes sobre cada um desses pontos é igual ao cálculo para o robô no espaço de configuração dados por (3.8) e (3.10). Cada um dos gradientes então é transferido para o espaço de configuração por (3.15) e todos são somados, gerando um gradiente resultante que é usado para produzir o sinal de controle dado por (3.1). Dessa forma, usamos as forças sobre esses pontos no espaço operacional para encontrar a força generalizada no espaço de configuração [1].

Para robôs poligonais, costuma-se utilizar os vértices como os pontos para os quais os gradientes são calculados, embora isso não seja obrigatório. Além disso, não há um número fixo de pontos a serem escolhidos, embora a quantidade de DoF determine a quantidade mínima de pontos. No Capítulo 2, vimos que para um corpo rígido no plano, são necessários pelo menos dois pontos.

Para esse trabalho foram utilizados os quatro vértices do retângulo para os potenciais repulsivos e os dois vértices da frente para o potencial atrativo (Figura 3.7).

Para cada ponto $r_j(q)$ escolhido e cada obstáculo i temos, de acordo com as equações anteriores, as seguintes expressões:

$$\nabla U_{att,j}(q) = \begin{cases} \xi_j\left(r_j(q) - r_j(q_{ref})\right), & d\left(r_j(q), r_j(q_{ref})\right) \le d^*_{ref} \\ \dfrac{d^*_{ref}\ \xi_j\left(r_j(q) - r_2(q_{ref})\right)}{d\left(r_j(q), r_j(q_{ref})\right)}, & d\left(r_j(q), r_j(q_{ref})\right) > d^*_{ref} \end{cases}$$

$$\nabla U_{rep_{i,j}}(q) = \begin{cases} \eta_j\left(\dfrac{1}{Q_i^{max}} - \dfrac{1}{d_i(r_j(q))}\right)\dfrac{\nabla d_i(r_j(q))}{d_i(r_j(q))^2}, & d_i(r_j(q)) \le Q_i^{max} \\ 0, & d_i(r_j(q)) > Q_i^{max} \end{cases}$$

$$d_i(r_j(q)) = \min_{c \in QO_i} d(r_j(q) - c) \qquad \text{e} \qquad \nabla d_i(r_j(q)) = \frac{r_j(q) - c}{d(r_j(q), c)}$$

As constantes ξ_j, η_j e Q_i^{max} costumam ser escolhidas empiricamente e não é preciso que elas sejam iguais para todos os pontos e obstáculos.

Então o sinal de controle considerado novamente como um sinal de velocidade, é dado por:

$$v = -\sum_j J_j^T(q)\nabla U_{att,j}(q) - \sum_j \sum_i J_j^T(\mathrm{q})\nabla U_{rep_{i,j}}(q).$$

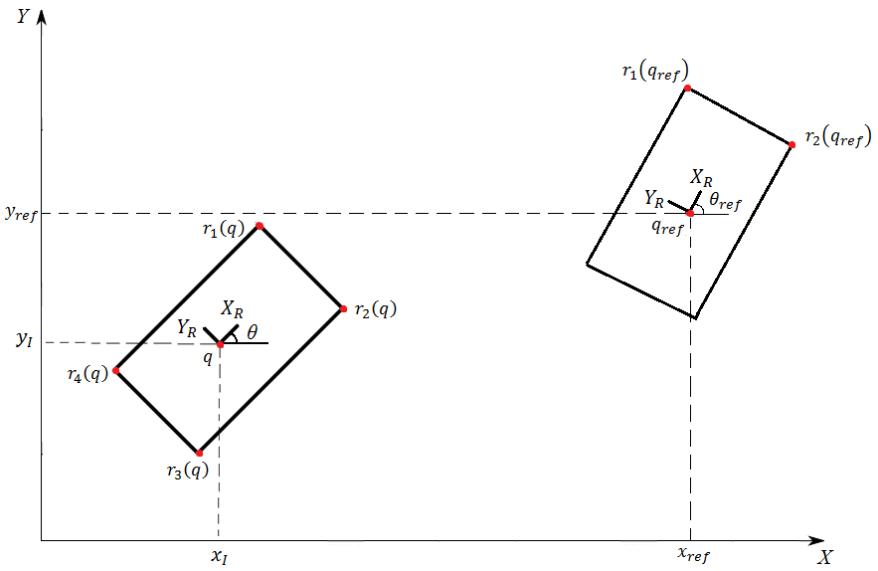


Figura 3.7: Pontos de referência do robô.

Capítulo 4
Localização

Para o cálculo do campo atrativo no APF, é necessário que o robô seja capaz de determinar sua própria configuração. Esse problema é denominado Localização. O algoritmo de localização usado nesse trabalho é o *Point-to-Line Iterative Closest Point* (PlICP) [7], que é uma variante do *Iterative Closest Point* (ICP) [8], um algoritmo de Alinhamento de Conjuntos de Pontos (*point set registration*).

Matematicamente, dois conjuntos P e X obtidos de uma mesma forma estão alinhados quando todos os pares de pontos (p_i, x_j), um de cada conjunto, coincidem depois que p_i sofre uma rotação $R(\theta)$ e uma translação t.

$$\forall p_i \in P, \exists x_j \in X \mid \left\| R(\theta)p_i + t - x_j \right\| = 0.$$

Entre os objetivos desse processo estão a fusão de múltiplos conjuntos de dados em um único modelo globalmente consistente, determinar se um conjunto de pontos corresponde a outro já conhecido ou, como no caso desse trabalho, mapear um novo conjunto de medidas dos sensores de um robô em relação a um conjunto anterior para estimar a sua configuração.

4.1 ICP

ICP é um método que permite o alinhamento preciso e computacionalmente eficaz de formas em 2D e 3D. Ele requer apenas um procedimento para achar a distância mínima entre um dado ponto e uma entidade geométrica qualquer (um ponto, uma curva, uma superfície, ou um conjunto de pontos, curvas e superfícies). ICP é um algoritmo de otimização que sempre converge monotonicamente para o mínimo local da função custo mais próximo da transformação espacial inicial [8]. A função custo é uma dada métrica de distância entre os dois conjuntos.

O objetivo do algoritmo consiste em, para um dado conjunto de pontos obtido por um sensor no sistema de coordenadas do sensor que pode corresponder a uma dada forma modelo, encontrar a rotação e translação ótimas que melhor alinham o conjunto de pontos ao modelo, minimizando a distância entre ambos. O algoritmo é computacionalmente eficiente, pois não envolve pré-processamento dos dados e não exige a derivação das equações de curvas ou superfícies.

Para calcular a distância mínima entre o conjunto de dados e o modelo, é preciso verificar a distância mínima entre cada ponto do conjunto e o modelo. Como trabalhamos com pontos obtidos pelo *laser scanner*, o modelo também é um conjunto de pontos, mas no caso geral o modelo poderia ser uma curva ou superfície (definidas parametricamente ou implicitamente), ou conjuntos de curvas e superfícies.
A métrica usada para calcular a distância entre dois pontos $\vec{p}_1$ e $\vec{p}_2$ é a distância Euclidiana $d(\vec{p}_1, \vec{p}_2)$.

Seja A o conjunto de N_a pontos $\vec{a}_i$: $A = \{\vec{a}_i\}$ para $i = 1, \dots, N_a$. A distância entre o ponto $\vec{p}$ e o conjunto A é:

$$d(\vec{p}, A) = \min_{i \in \{1,\dots,N_a\}} d(\vec{p}, \vec{a}_i) \ ,$$

onde o ponto $\vec{a}_i$ satisfaz a igualdade $d(\vec{p}, \vec{a}_i) = d(\vec{p}, A)$, ilustrada na Figura 4.1.

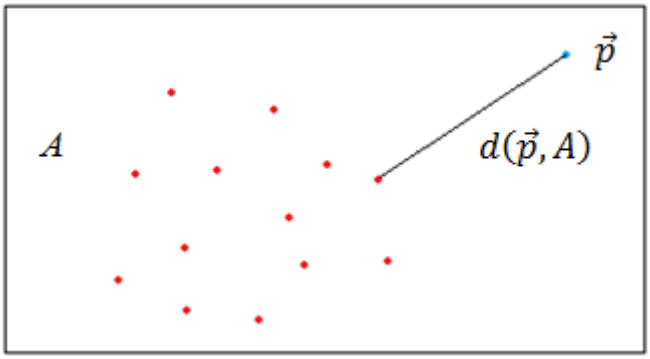


Figura 4.1: Distância entre um ponto e um conjunto de pontos.

Tendo as distâncias de cada ponto do conjunto de dados *P* até o modelo *X*, com o mesmo número de pontos $N_P = N_X$, podemos agora calcular a rotação e a translação ótimas para minimizar a distância entre *P* e *X*. Sem perda de generalidade, assumimos que o ponto mais próximo de p_i em *X* é x_i, com o mesmo índice.

O método usa a representação em quaternion $\vec{q}_R = [q_0\ q_1\ q_2\ q_3]^T$ para a rotação, sendo $q_0 \geq 0$ e $q_0^2 + q_1^2 + q_2^2 + q_3^2 = 1$. A matriz de rotação em função do quaternion é dada por (2.11) e $\vec{t} = [t_1\ t_2\ t_3\ ]^T$ é o vetor de translação. O vetor completo de alinhamento é $\vec{r} = \left[\vec{q}_R \middle| \vec{t}\right]^T$. *P'* é o conjunto de pontos de *P* após serem transformados por $\vec{r}$, ou seja, para cada $\vec{p}_i$ em *P*, $\overrightarrow{p'}_i = R(\vec{q}_R)\vec{p}_i + \vec{t}$. Idealmente, considerando que não existem incertezas no sensor, haveria um vetor $\vec{r}$ tal que o conjunto *P'* coincidiria com *X*. Como as incertezas existem, esse pareamento entre *P'* e *X* não é perfeito, então devemos buscar um vetor $\vec{r}$ que produza a menor soma dos erros entre os pontos dos dois conjuntos. A função objetivo a ser minimizada, portanto, é o erro quadrático médio (*mean quadratic error*, MSE) $f(\vec{r})$ de todos os pontos de *P'* em relação aos seus pontos mais próximos em *X*:

$$f(\vec{r}) = \frac{1}{N_P}\sum_{i=1}^{N_P}\left\|\vec{x}_i - R(\vec{q}_R)\vec{p}_i - \vec{t}\right\|^2, \tag{4.1}$$

$$r_{min} = \min_{\vec{r}} f(\vec{r}). \tag{4.2}$$

A escolha dessa métrica de erro para a função objetivo fez com que esse método ICP original passasse a ser chamado *point-to-point ICP*. Muitos algoritmos variantes do método usam uma métrica *point-to-line* ou *point-to-plane* como visto à frente.

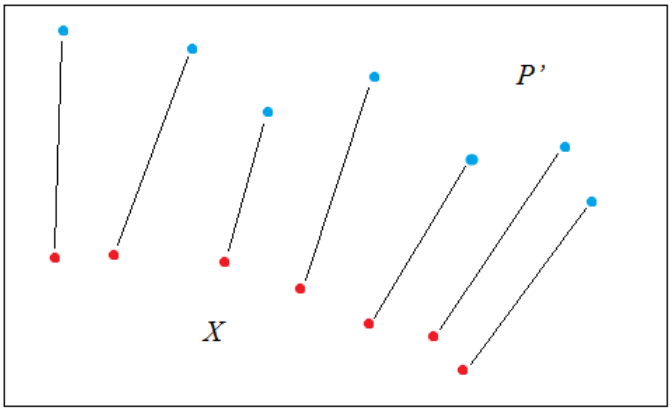


Figura 4.2: Métrica *point-to-point*.

Existem soluções fechada para essa minimização, como a solução abaixo, proposta em [9], que foi a solução utilizada originalmente para o ICP. Foge ao escopo desse trabalho demonstrar o desenvolvimento completo dessa solução, portanto apenas a resposta final é apresentada abaixo:

Os centróides dos conjuntos são definidos como:

$$\overrightarrow{\mu}_p = \frac{1}{N_p}\sum_{i=1}^{N_p} \vec{p}_i \quad , \qquad \vec{\mu}_x = \frac{1}{N_x}\sum_{i=1}^{N_x} \vec{x}_i \ . \tag{4.3}$$

A matriz de covariância cruzada de *P* e *X* é dada por:

$$\Sigma_{px} = \frac{1}{N_p}\sum_{i=1}^{N_p}[(\vec{p}_i - \overrightarrow{\mu}_p)(\vec{x}_i - \vec{\mu}_x)^T] \ = \frac{1}{N_p}\sum_{i=1}^{N_p}[\vec{p}_i\vec{x}_i^T] - \overrightarrow{\mu}_p\vec{\mu}_x^T. \tag{4.4}$$

Os componentes dessa matriz são usados para determinar o vetor-coluna Δ:

$$\Delta = \left[\left(\Sigma_{px} - \Sigma_{px}^T\right)_{23} \quad \left(\Sigma_{px} - \Sigma_{px}^T\right)_{31} \quad \left(\Sigma_{px} - \Sigma_{px}^T\right)_{12}\right]^T. \tag{4.5}$$

A matriz simétrica $Q(\Sigma_{px})$ é:

$$Q(\Sigma_{px}) = \begin{bmatrix} tr(\Sigma_{px}) & \Delta^T \\ \Delta & \Sigma_{px} + \Sigma_{px}^T - tr(\Sigma_{px})I_3 \end{bmatrix}. \tag{4.6}$$

E $\vec{q}_R$ é o autovetor associado ao maior autovalor dessa matriz $Q(\Sigma_{px})$ e

$$\vec{t} = \vec{\mu}_x - R(\vec{q}_R)\vec{\mu}_p. \tag{4.7}$$

Essa operação com quaternions de (4.3) a (4.7) é denotada por:

$$(\vec{r}, d_{ms}) = R(P, X),$$

onde $d_{ms} = f(\vec{r})$.

4.1.1 Algoritmo ICP

Foi assumido anteriormente que $N_x = N_p$ para simplificar a explicação, mas agora essa premissa não será necessária, pois usaremos no algoritmo um subconjunto *Y* de *X*, que consiste de todos pontos que são os mais próximos de algum ponto em *P*, uma vez que a igualdade $N_y = N_p$ é sempre satisfeita. Essa operação é denotada por:

$$Y = C(P, X).$$

O conjunto *P* pode ser qualquer entidade geométrica, mas para o algoritmo é necessário que ele seja transformado em um conjunto de pontos. Por exemplo, se *P* é um conjunto de segmentos de retas, escolhemos os pontos das extremidades de cada segmento. Se for uma curva, devemos aproximá-la por uma sequência de segmentos de reta e escolher as extremidades.
O vetor de alinhamento $\vec{r}$ é dado por:

$$(\vec{r}, d) = R(P, Y),$$

e a posição dos pontos em *P* é atualizada a cada iteração por $P = \vec{r}(P)$, que significa $\vec{p}_i = R(\vec{q}_R)\vec{p}_i + \vec{t}$, para cada $\vec{p}_i$.

Note que *P* é o conjunto atual de pontos obtidos pelo *laser scanner* e o modelo *X* é o conjunto anterior obtido *pelo laser scanner*.

Uma descrição resumida do funcionamento do algoritmo é apresentada a seguir:

Inputs: Conjuntos de dados *P* e *X*, tolerância τ para o MSE entre duas iterações e transformação inicial $\vec{r}_0$.

- $P_0 = P, k = 0$
- Enquanto $\left((d_k - d_{k+1}) > \tau\right)$
    - $Y_k = C(P_k, X)$
    - $(\vec{r}_k, d_k) = R(P_0, Y_k)$
    - $P_{k+1} = \vec{r}_k(P_0)$

A cada iteração, o vetor $\vec{r}_k$ é definido em relação a P_0 para que o vetor final represente a transformação completa. Caso fosse definido em relação à P_k ainda seria necessário fazer a multiplicação entre quaternions, gerando cálculos desnecessários.

A transformação inicial identidade $\vec{r}_0 = [1\ 0\ 0\ 0\ 0\ 0\ 0]^T$ é usada quando não há uma boa estimativa para esse valor inicial. Para a localização de um robô móvel, porém, um bom valor para $\vec{r}_0$ é a transformação $\vec{r}$ encontrada pela execução do algoritmo para encontrar a localização do robô no instante anterior, já que a velocidade do robô é tipicamente muito menor do que a frequência com que o *laser* obtém novos conjuntos de pontos e, por esse motivo, a variação de posição e orientação entre dois conjuntos consecutivos é pequena.

4.1.2 Teorema da Convergência

A convergência do ICP pode ser assegurada a partir do seguinte teorema:

Teorema: O algoritmo ICP sempre converge monotonicamente para um mínimo local do erro quadrático médio.

Prova: Dados P_k e X, é computado o conjunto Y_k.
O erro médio quadrático da correspondência é dado por:

$$e_k = \frac{1}{N_P}\sum_{i=1}^{N_P}\|\vec{y}_{ik} - \vec{p}_{ik}\|^2.$$

Após o alinhamento, temos o MSE d_k, descrito por:

$$d_k = \frac{1}{N_P}\sum_{i=1}^{N_P}\left\|\vec{y}_{i,k} - R(\vec{q}_{kR})\vec{p}_{i,0} - \vec{t}_k\right\|^2.$$

Para provar que $d_k \leq e_k$, vamos negar essa afirmação e supor que $d_k > e_k$. Nesse caso, a transformação identidade $\vec{r}_{kI} = [1\ 0\ 0\ 0\ 0\ 0\ 0]^T$ em *P* teria produzido $d_{kI} = e_k < d_k$, e haveria um valor $\vec{r}_{kI}$ para o qual o MSE seria menor do que aquele encontrado pelo método de minimização, o que é uma contradição. Com isso, verificamos que $d_k \leq e_k$.

Agora fazemos $P_{k+1} = \vec{r}_k(P_0)$. Se o conjunto anterior de correspondências Y_k fosse mantido o valor de d_k ainda seria:

$$d_k = \frac{1}{N_P}\sum_{i=1}^{N_P}\left\|\vec{y}_{i,k} - \vec{p}_{i,k+1}\right\|^2.$$

Contudo, quando um novo $Y_{k+1} = C(P_{k+1}, X)$ é obtido, verificamos que $\left\|\vec{y}_{i,k+1} - \vec{p}_{i,k+1}\right\| \leq \left\|\vec{y}_{i,k} - \vec{p}_{i,k+1}\right\|$ para cada *i*, ou $e_{k+1} \leq d_k$, porque o ponto $\vec{y}_{i,k}$ era o ponto mais próximo antes do alinhamento e agora está a uma nova distância de $\vec{p}_{i,k+1}$. Se $\vec{y}_{i,k+1}$ estivesse mais distante de $\vec{p}_{i,k+1}$ do que $\vec{y}_{i,k}$, isso iria contradizer a operação $Y_{k+1} = C(P_{k+1}, X)$. Com isso, as seguintes relações são válidas para toda iteração *k*:

$$0 \leq d_{k+1} \leq e_{k+1} \leq d_k \leq e_k,$$

onde o limite inferior ocorre porque o erro quadrático não pode ser negativo para números reais. Como a sequência de erros quadráticos não pode crescer e possui um limitante inferior, o algoritmo converge monotonicamente para um valor mínimo. Empiricamente, verifica-se que essa convergência é linear [10].

4.2 Variantes do ICP

Desde a introdução do ICP, muitas variantes foram criadas para melhorar a velocidade de convergência, a acurácia da resposta final, a capacidade de encontrar a resposta correta para modelos geometricamente mais complexos e a robustez do algoritmo original [10]. Para cada estágio do algoritmo existe pelo menos uma variante que a altera. De acordo com isso, podemos classificar as variantes em relação ao modo como afetam cada estágio, com alguns exemplos de variações que podem ocorrer em cada estágio:

1-Seleção de pontos a serem testados em um ou nos dois conjuntos *P* e *X*.

- Todos os pontos são testados;
- Amostragem aleatória de pontos a serem testados;
- Amostragem uniforme;
- Amostragem baseada em extração de *features* (características do modelo que são essenciais para o alinhamenhto correto).

2-Pareamento dos pontos de um conjunto em relação ao outro.

- Para cada ponto em P, encontrar o mais próximo em *X;*
- "*Normal Shooting*" (encontrar a interseção entre a reta normal que se origina em um ponto de *P* e o modelo *X*);
- "*Reverse Calibration*" (Projeção do ponto de *P* em *X*. Esse método de pareamento é menos sensivel a ruídos e *outliers*).

3-Ponderação apropriada dos pares de pontos correspondentes na função de minimização.

- Pesos iguais para todos os pares;
- Pesos menores para pares cuja distância ponto-a-ponto seja maior;
- Pesos baseados na incerteza dos pontos dos pares.

4-Rejeição de certos pares baseada na análise individual de cada par ou na análise do conjunto completo de pares.

- rejeição de pontos que estejam nas extremidades de um dos conjuntos (útil quando os dois conjuntos não tem correspondência perfeita, o que ocorre na maioria dos casos);
- rejeição de uma certa fração de pares que tenham a maior distância ponto-a-ponto;
- rejeição de pares que tenham distância ponto-a-ponto maior que uma tolerância.

5-Escolha da métrica de erro a ser minimizada baseada nos pares de pontos.

- *point-to-point*: soma dos quadrados das distâncias ponto-a-ponto dos pares de correspondentes;
- *point-to-line*: soma dos quadrados das distâncias perpendiculares de cada ponto em *P* até o segmento de reta entre os seus dois pontos mais próximos em *X*;
- *point-to-plane*: soma dos quadrados das distâncias perpendiculares de cada ponto em *P* até o plano tangente à *X* que contém o ponto correspondente.

6-Minimização da métrica de erro.

- Para o *point-to-point* existem soluções em forma fechada para transformações rígidas que minimizam o erro, como a do quaternion, mostrada acima, e por Decomposição em Valores Singulares (SVD), entre outras;
- Para o *point-to-line* existem soluções em forma fechada para transformações rígidas que minimizam o erro, caso os conjuntos de dados estejam no plano;
- Para o *point-to-plane* não existem soluções em forma fechada. As equações de mínimos quadrados são resolvidas por métodos não-lineares clássicos como Levenberg-Marquardt [11]. Embora empiricamente [10] se verifique que cada iteração de um algoritmo *point-to-plane* é mais lenta que a de um algoritmo *point-to-point*, tambem se verifica que a sua taxa de convergência é melhor. Algumas variantes fazem as aproximações lineares $sin(\theta) \cong \theta$ e $\cos(\theta) \cong 1$ nas matrizes de rotação, assumindo que as rotações são pequenas, e resolvem o problema usando métodos lineares clássicos [12].

4.3 PlICP

A variante do ICP implementada nesse trabalho é o *Point-to-Line Iterative Closest Point* (PlICP). Como indicado pelo nome, esse algoritmo usa uma métrica diferente para o problema de mínimos quadrados [13], para a qual também há uma solução fechada. O algoritmo converge quadraticamente para a solução [14] e em um número finito de iterações se o modelo *X* for uma cadeia poligonal, uma sequência de segmentos de reta conectados [7]. Para esse trabalho, essa condição é sempre verdadeira.

Mais uma vez, seja *P* o conjunto de pontos $\{p_i\}$ e *X* o modelo com o qual se deseja fazer o alinhamento de *P*, que contém os pontos $\{x_i\}$.

Seja $\vec{r} = \left[\vec{t}\ \theta\right]^T$ o vetor de alinhamento, sendo $\vec{t}$ o vetor de translação e θ o ângulo de rotação. Então, $\vec{p}_i^{\,w}$ é o valor de $\vec{p}_i$ após o alinhamento, ou seja,

$$\vec{p}_i^{\,w} = R(\theta)\vec{p}_i + \vec{t}\,, \tag{4.8}$$

onde $\vec{n}_i$ é a reta que projeta $\vec{p}_i^{\,w}$ em *X*, sendo portanto normal a *X*.

Os dois pontos de *X* mais próximos de $\vec{p}_i^{\,w}$ são denotados $\vec{p}_{\,j_1^i}$ e $\vec{p}_{\,j_2^i}$.

Com essas definições, o objetivo na métrica *point-to-line* é dado por:

$$\min_{\vec{r}} \sum_i \left(n_i^T \left[R(\theta)\vec{p}_i + \vec{t} - \vec{p}_{\,j_1^i}\right]\right)^2, \tag{4.9}$$

onde o termo $n_i^T \left[R(\theta)\vec{p}_i + \vec{t} - \vec{p}_{\,j_1^i}\right]$ é um produto escalar que representa a projeção da distância entre $\vec{p}_i^{\,w}$ e um de seus pontos mais próximos em relação à reta normal a *X*, como mostrado na Figura 4.3.

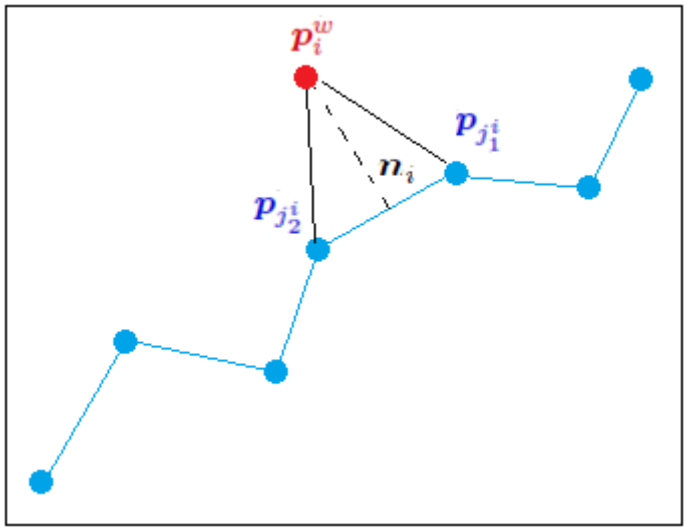


Figura 4.3: Métrica *point-to-line.*

A solução fechada desse problema é dada a seguir [7]:

A solução para $\vec{r}$ é dada na forma

$$\vec{z} = [z_1\ z_2\ z_3\ z_4]^T = \left[t_x\ t_y \cos(\theta) \sin(\theta)\right]^T,$$

com a restrição $$z_3^2 + z_4^2 = 1. \tag{4.10}$$

Seja M_i definida como a matriz:

$$M_i = \begin{bmatrix} 1 & 0 & p_{i0} & -p_{i1} \\ 0 & 1 & p_{i1} & p_{i0} \end{bmatrix}, \tag{4.11}$$

onde $\vec{p}_i = [p_{i0}\ p_{i1}]^T$.

Então (4.9) pode ser reescrita como:

$$\min_{\vec{z}} \sum_i \left(M_i\vec{z} - \vec{p}_{\,j_1^i}\right)^T \vec{n}_i\vec{n}_i^T \left(M_i\vec{z} - \vec{p}_{\,j_1^i}\right). \tag{4.12}$$

Expandindo (4.12) temos:

$$\sum_i \left(M_i\vec{z} - \vec{p}_{j_1^i}\right)^T \vec{n}_i\vec{n}_i^T \left(M_i\vec{z} - \vec{p}_{j_1^i}\right),$$

$$\sum_i \left(\vec{z}^T M_i^T \vec{n}_i\vec{n}_i^T M_i\vec{z} + \vec{p}_{j_1^i}^T \vec{n}_i\vec{n}_i^T \vec{p}_{j_1^i} - 2\vec{p}_{j_1^i}^T \vec{n}_i\vec{n}_i^T M_i\vec{z}\right),$$

e ignorando o termo constante do meio, que não entra na minimização:

$$\vec{z}^T(\sum_i M_i^T \vec{n}_i\vec{n}_i^T M_i\ )\vec{z}\ - \left(\sum_i 2\vec{p}_{j_1^i}^T \vec{n}_i\vec{n}_i^T M_i\right)\vec{z}. \tag{4.13}$$

Fazendo $M = (\sum_i M_i^T \vec{n}_i\vec{n}_i^T M_i\ )$ e $G = \left(-\sum_i 2\vec{p}_{j_1^i}^T \vec{n}_i\vec{n}_i^T M_i\right)^T$ obtém-se:

$$\vec{z}^T M\vec{z}\ + G^T\vec{z}.$$

Definindo a matriz esparsa W como:

$$W = \begin{bmatrix} 0_{2\times2} & 0_{2\times2} \\ 0_{2\times2} & I_{2\times2} \end{bmatrix},$$

a restrição (4.10) pode ser reescrita como $\vec{z}^T W\vec{z} = 1$.

Com isso, o problema de minimização sem restrição (4.9) pode ser reescrito na forma quadrática com restrição de igualdade:

$$\min_{\vec{z}}\ \vec{z}^T M\vec{z}\ + G^T\vec{z}, \tag{4.14}$$

$$\text{sujeito a} \qquad \vec{z}^T W\vec{z} - 1 = 0. \tag{4.15}$$

Nessa forma, o problema pode ser resolvido por Multiplicadores de Lagrange:

Definimos a função:

$$L(\vec{z}) = \vec{z}^T M\vec{z}\ + G^T\vec{z} + \lambda(\vec{z}^T W\vec{z} - 1), \tag{4.16}$$

Seus pontos críticos são encontrados fazendo $\frac{\partial L}{\partial \vec{z}} = 0^T$, isto é:

$$\frac{\partial L}{\partial \vec{z}} = 2\vec{z}^T M + G^T + 2\lambda \vec{z}^T W = 0^T, \tag{4.17}$$

e isolando $\vec{z}$, tem-se:

$$\vec{z} = -(2M + 2\lambda W)^{-1} G. \tag{4.18}$$

Substituindo (4.18) em (4.15) temos:

$$G^T (2M + 2\lambda W)^{-1} W (2M + 2\lambda W)^{-1} G = 1, \tag{4.19}$$

que é uma equação polinomial de quarta ordem em λ. Qualquer método para solucionar equações polinomiais pode ser usado para achar λ e esse valor é substituído em (4.18) para encontrar $\vec{z}$. Uma descrição resumida do algoritmo é apresentada a seguir:

Inputs: O último e o penúltimo conjuntos de pontos gerados pelo *laser scanner* y_t e y_{t-1}, respectivamente; um vetor de alinhamento inicial $\vec{r}_0$, tolerância τ e $k = 0$.

- $P = y_t$
- X é criado a partir de y_{t-1}, sendo a sequência de segmentos de reta que unem os pontos suficientemente próximos de y_{t-1} (cadeia poligonal)
- Enquanto $\left(\left(d(\vec{r}_k, C_{k-1}) - d(\vec{r}_{k+1}, C_k) \right) > \tau \right)$
  - $\vec{p}_i^{\,w} = R(\theta_k)\vec{p}_i + \vec{t}_k$
  - Para cada $\vec{p}_i^{\,w}$ encontrar $\vec{p}_{\,j_1^i{}_k}$ e $\vec{p}_{\,j_2^i{}_k}$
  - Criar o conjunto de tuplas C_k. Cada elemento de C_k é $\left(i, j_1^i, j_2^i\right)$, que significa que $\vec{p}_i^{\,w}$ está pareado com o segmento $\vec{p}_{\,j_1^i{}_k} - \vec{p}_{\,j_2^i{}_k}$
  - $d(\vec{r}_{k+1}, C_k) = \sum_i \left(n_{i\ k}^T \left[R(\theta_{k+1})\vec{p}_i + \vec{t}_{k+1} - \vec{p}_{\,j_1^i{}_k} \right] \right)^2$
  - Obter $\vec{r}_{k+1}$ a partir de (4.18)

Capítulo 5
Extração de Linhas

Para o programa de planejamento de trajetória implementado, são necessários somente o algoritmo APF descrito no Capítulo 3 e o algoritmo PlICp descrito no Capítulo 4. Contudo, foi implementado outro programa para mostrar uma aplicação prática do planejamento de trajetória para portadores de deficiência física.

O objetivo é que a referência de configuração dada ao APF como entrada seja determinada automaticamente por um outro programa, que por sua vez recebe comandos simples de alguém que está na cadeira de rodas através de um *joystick* ou por ondas cerebrais lidas por eletroencefalograma (EEG). Esses comandos simples são: seguir em frente, parar, ou virar a cadeira no próximo corredor à esquerda ou à direita. Para determinar onde está o próximo corredor, não basta ter acesso ao conjunto de pontos obtido pelo *laser scanner*. É preciso determinar as retas que compõem os corredores a partir desses pontos. Para isso, é necessário utilizar um algoritmo de extração de linhas e o algoritmo usado nesse trabalho é o *Random Sample Consensus* (RANSAC) [15].

5.1 Extração de Características

Um robô móvel deve ser capaz de determinar sua relação com o ambiente ao seu redor através de medições feitas pelos seus sensores e, com base nessa relação, tomar ações que permitam atingir seu objetivo. Contudo, como todo sensor possui incertezas, as medições não devem ser utilizadas diretamente.

É comum fazer uma filtragem dos dados para eliminar aqueles que sejam muito ruidosos e que prejudicariam de forma significativa as ações do robô.

A extração de características (*feature extraction*) é o processo de extrair informações tratadas de um ou mais sensores para gerar uma representação em nível mais alto das medições, como um mapa, por exemplo, que o robô seja capaz de utilizar para decidir suas ações futuras. Isso é feito comparando e pareando medições do sensor com modelos pré-definidos.

Por ser um processo de nível mais alto cujos algoritmos exigem alto esforço computacional, a extração de características não precisa ser implementada para qualquer tarefa que o robô deva executar. Tarefas simples, como a parada de emergência dos motores de um veículo diante de um obstáculo próximo podem ser executadas com os dados brutos de um sensor, já para tarefas mais sofisticadas, como o mapeamento de uma região ou navegação de alta precisão esse processo se torna fundamental.

De acordo com [4], uma característica deve ser uma estrutura comum dos elementos no ambiente que seja facilmente reconhecida e extraída das medições do sensor e que possa ser matematicamente descrita. Podem ser figuras simples como primitivas geométricas (retas e círculos) ou mais complexas como portas, cadeiras, árvores ou a digital de um dedo.

O tipo de característica buscada em uma aplicação depende dos tipos de sensores disponíveis e do ambiente em que o robô vai atuar. Para o conjunto de corredores no qual a cadeira de rodas se desloca, retas são características que definem com alta precisão o ambiente e são facilmente identificáveis com o *laser scanner*.

De fato, na maioria dos casos, as características buscadas com sensores de distância como o *laser* são segmentos de retas e elipses. Isso ocorre pois a extração é feita comparando e pareando medições do sensor com as descrições matemáticas. Como normalmente o sistema é superdeterminado (existem mais pontos medidos pelo sensor do que parâmetros da descrição), não existe solução perfeitamente consistente devido às incertezas dos pontos do *laser*. O problema de extração, portanto, é de otimização (minimização de erro). Para figuras simples como a reta e o círculo existem soluções fechadas para esse problema. Para outras figuras, a descrição matemática é muito complexa e soluções fechadas para identificá-las ainda não foram encontradas e os métodos iterativos usados consomem muito tempo do computador.

Para a extração de retas, especificamente, uma solução seria usar diretamente o Método dos Mínimos Quadrados (MMQ) no conjunto de pontos. Infelizmente, isso funciona somente nas situações mais simples, pois normalmente há mais de uma reta a ser encontrada no conjunto e, além disso, o MMQ é muito sensível a ruídos. Pontos muito distantes da reta real provocam erros significativos nos parâmetros da reta encontrada pelo MMQ. Ainda assim, alguns algoritmos usam o MMQ internamente, tomando precauções para evitar suas desvantagens, embora o RANSAC não o use. Um algoritmo de extração de retas robusto deve ser capaz de identificar qualquer número de retas, determinando que pontos do conjunto devem pertencer a que retas (segmentação

do conjunto de pontos), calcular os parâmetros da melhor reta para cada subconjunto e ser capaz de ignorar ruídos (robustez).

5.2 RANSAC

É um algoritmo muito robusto, capaz de extrair primitivas geométricas mesmo na presença de grandes quantidades de *outliers*. *Outliers* são pontos que não se adequam ao modelo da primitiva que está sendo buscada, sendo, portanto, pontos que pertencem a objetos que não são do tipo da primitiva ou são provocados por ruído excessivo. Em uma sala, por exemplo, podemos procurar pontos que pertençam às paredes, mas também haverá pontos pertencentes a outros corpos, como cadeiras e pessoas, que não se ajustam às retas. Pontos que se encaixem na primitiva buscada são chamados de *inliers*.

O RANSAC é iterativo e probabilístico, no sentido que a probabilidade de encontrar uma reta livre de *outliers* aumenta conforme cresce o número de iterações usadas e funciona de forma muito eficiente para extração de retas, planos e elipses.

O algoritmo é iniciado recebendo como entradas o modelo da primitiva a ser extraída, o conjunto de pontos A, uma tolerância *d* e a probabilidade desejada *p* de ser encontrada a melhor reta. A cada iteração, RANSAC escolhe aleatoriamente *n* pontos de A, em que *n* é o número de parâmetros do modelo da primitiva. Os *n* parâmetros do modelo são determinados para os *n* pontos escolhidos (sistema determinado com solução única). Para cada ponto de A é verificada a distância mínima até a primitiva encontrada. Se a distância é menor que *d*, o ponto faz parte do conjunto de *inliers* dessa primitiva. O algoritmo guarda a quantidade de *inliers* correspondente ao conjunto de parâmetros e inicia outra iteração. Quando todas as iterações terminam, a primitiva escolhida é aquela para cujo conjunto de parâmetros foi encontrado o maior número de *inliers*.

Se a primitiva for uma reta especificamente o modelo é $y = ax + b$ e portanto $n = 2$. A cada iteração, o RANSAC escolhe dois pontos aleatoriamente, calcula os parâmetros *a* e *b* da reta entre eles e verifica a distância perpendicular entre todos os outros pontos de A e a reta encontrada. O processo é ilustrado na Figura 5.1.

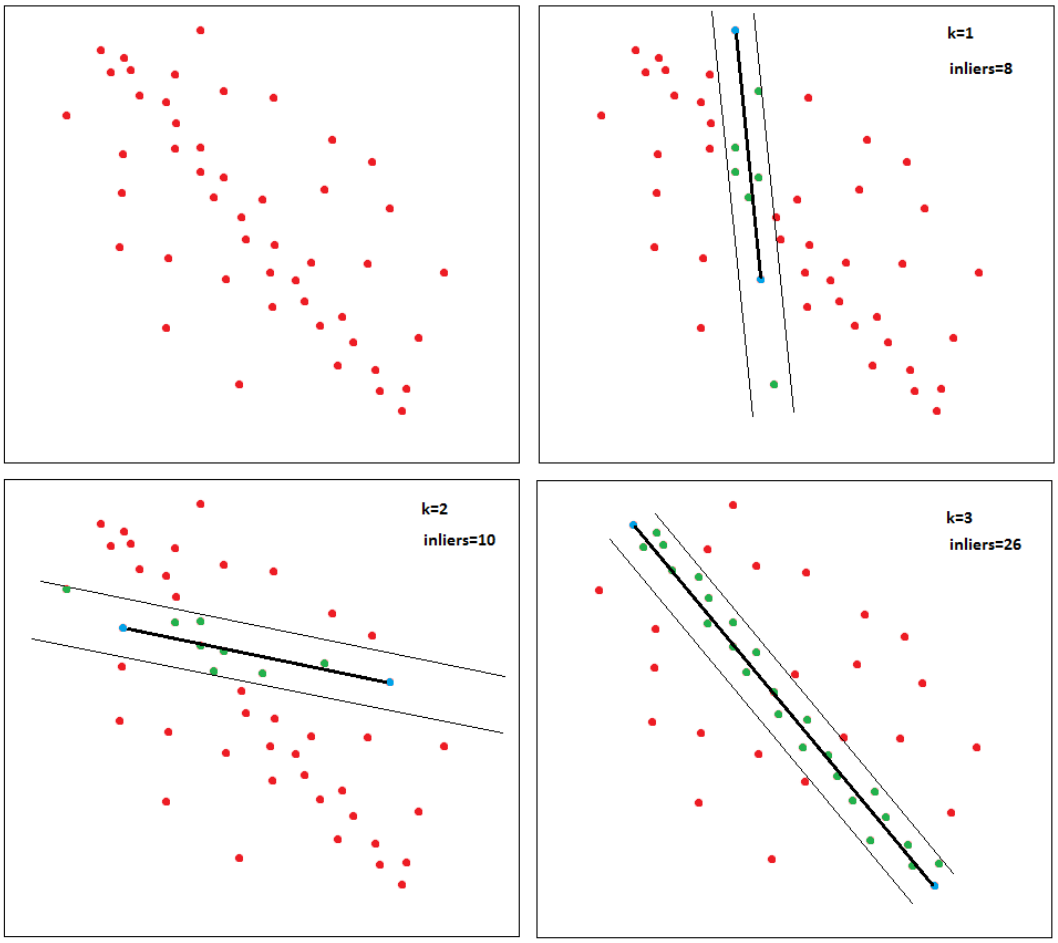


Figura 5.1: Iterações do RANSAC.

A questão é quantas iterações devem ser executadas. A escolha mais natural seria testar todas as possibilidades, e dessa forma o RANSAC seria determinístico. O problema é que os sensores normalmente produzem um número de pontos muito grande (um *laser scanner* típico capta algumas centenas de pontos). Seja N o número de pontos em A, o total de possibilidades é $\frac{N(N-1)}{2}$. Para o *laser* usado na cadeira de rodas, $N = 541$, portanto são 146.070 possibilidades. É computacionalmente inviável testar todas essas possibilidades.

O RANSAC parte do princípio que nem todas as possibilidades precisam ser testadas. Com apenas uma pequena parcela delas é possível ter uma grande probabilidade de acerto se tivermos uma estimativa razoável da fração de *inliers* da melhor primitiva no conjunto A.

Seja p a probabilidade do RANSAC selecionar em alguma iteração apenas *inliers* da melhor primitiva possível no conjunto A quando escolhe os n pontos para estimar a

primitiva e *w* a probabilidade de escolher um *inlier* qualquer dessa melhor primitiva em A cada vez que um ponto é selecionado, ou seja, *w* é a fração de *inliers* em A. Como os pontos são escolhidos aleatoriamente, podemos assumir que as escolhas são eventos independentes. Com isso, a probabilidade de escolher *n* pontos que sejam *inliers* é w^n. Seu complemento $1 - w^n$ é a probabilidade de pelo menos um dos *n* pontos ser *outlier*, o que significa que um modelo ruim será obtido desse conjunto de *n* pontos. A probabilidade do algoritmo nunca encontrar *n inliers* em qualquer das *k* iterações é $(1 - w^n)^k$ e deve ser igual a $1 - p$:

$$1 - p = (1 - w^n)^k \,, \tag{5.1}$$

e isolando *k*, temos:

$$k = \frac{\log(1-p)}{\log(1-w^n)} \,, \tag{5.2}$$

e com isso podemos estimar o número de iterações *k* que o RANSAC precisa executar para ter probabilidade *p* de encontrar a melhor primitiva possível considerando que a fração de *inliers* seja *w*. Por exemplo, para encontrar a melhor reta para $N = 541$, $p = 0{,}99$ e $w = 0{,}5$, o número iterações é $k = 16$.

Com 16 iterações temos 99% de chances de encontrar a melhor reta, contra as 146.070 iterações do método determinístico.

É importante ressaltar que o número de iterações obtido por (5.2) não depende de *N*, mas o total de possibilidades sim. Quanto maior o número de pontos obtidos pelo sensor, maior a diferença de desempenho entre o RANSAC e o método determinístico.
O valor de *w* é uma entrada do algoritmo e não precisa ser um chute muito preciso. Algumas variantes do algoritmo calculam *w* a partir da tolerância *d*, mas isso não foi usado nesse trabalho.

Por último, embora o RANSAC tenha muitas vantagens, ele também apresenta desvantagens:

- Por ser probabilístico ele não é ótimo. É possível melhorar sua probabilidade de encontrar a melhor resposta aumentando o número de iterações, mas sempre haverá uma pequena probabilidade de ele não encontrar a melhor resposta, a não ser que sejam testadas todas as possibilidades, mas nesse caso o ganho de desempenho computacional devido ao número reduzido de iterações é perdido.
- O algoritmo é capaz de extrair diversos modelos geométricos, como retas e círculos, mas é capaz de extrair somente um modelo para cada execução, ou seja, ele não pode extrair retas e círculos simultaneamente.

Para esse trabalho nenhuma dessas desvantagens gera problemas. Para a primeira, mesmo na possibilidade remota que a reta encontrada não seja a melhor, ela tende a ter parâmetros muito próximos da melhor, e para a aplicação implementada, uma precisão tão alta não é necessária. A segunda desvantagem não tem nenhuma importância, pois nesse trabalho queremos encontrar apenas retas.

5.3 RANSAC para múltiplas retas

Caso seja necessário encontrar mais de uma reta, o RANSAC pode ser usado iterativamente. A cada reta encontrada, o seu conjunto de *inliers* é descartado do conjunto A que será usado pelo RANSAC para obter a próxima reta. Os critérios de parada nesse caso são:

1-Quando a reta obtida tem menos que um número mínimo de *inliers*. Esse número é um parâmetro fornecido pelo usuário.

2-Quando o número de pontos em A é menor que uma certa fração do número de pontos do conjunto A original. Essa fração é um parâmetro fornecido pelo usuário.

O critério usado nesse trabalho é o primeiro e com isso temos o seguinte algoritmo para extração de linhas:

Inputs: Conjunto de pontos A obtido pelo laser; tolerância *d*; fração de inliers *w*; número mínimo de *inliers n_min*

- Enquanto (*n_inliers* > *n_min*)
  - calcular *k* por (5.2)
  - $i = 0$
  - Enquanto ($i < k$)
    - selecionar 2 pontos de A aleatoriamente
    - calcular os parâmetros *a* e *b* da reta
    - calcular a distância perpendicular de todos os outros pontos de A até a reta. Para cada ponto, se a distância for menor que *d*, acrescentar esse ponto ao conjunto de *inliers* da iteração
  - escolher os parâmetros *a* e *b* da iteração com o maior número de *inliers*
  - eliminar os inliers da reta obtida do conjunto A

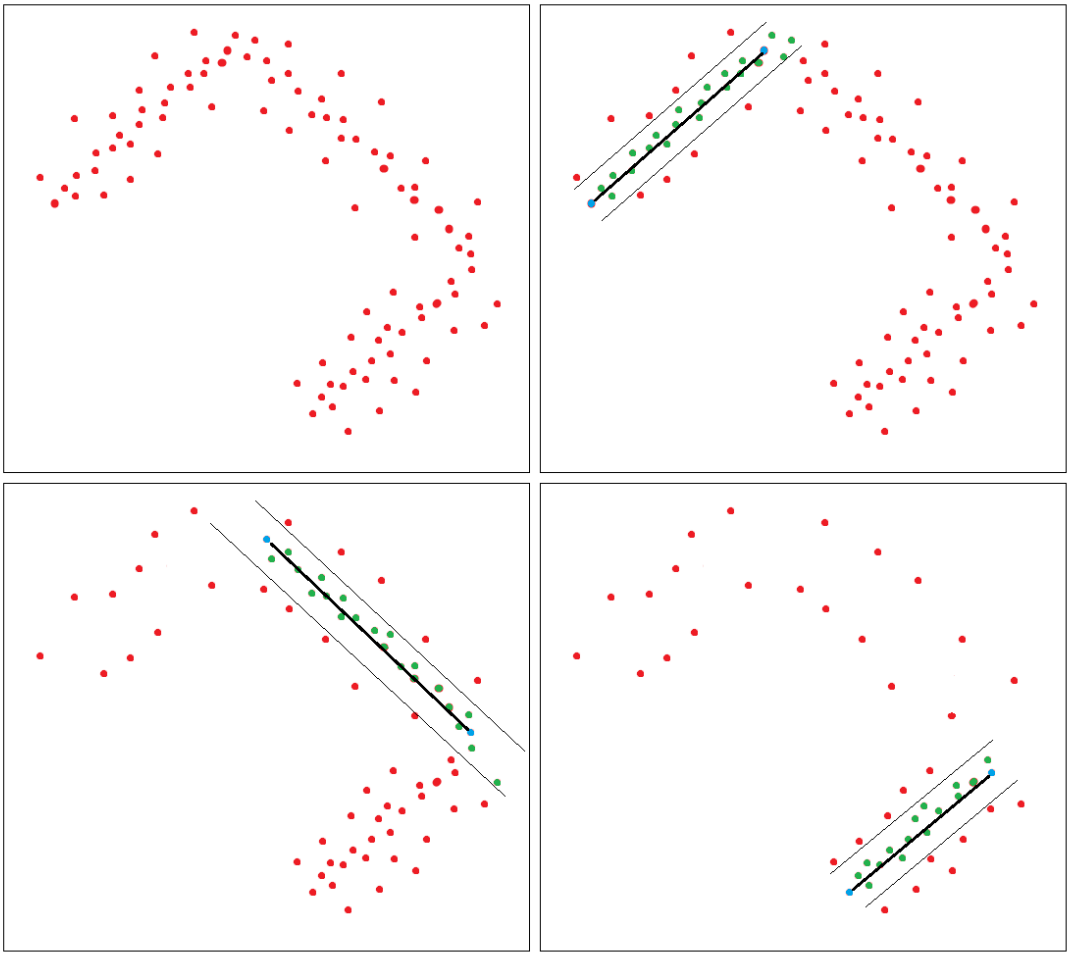


Figura 5.2: RANSAC para múltiplas retas.

Capítulo 6

Implementação

Nesse capítulo são descritos todos os equipamentos utilizados para a implementação do algoritmo, assim como o *software* utilizado. Também são apresentados os códigos desenvolvidos para o trabalho, seus resultados e conclusões. As especificações dos equipamentos usados na implementação estão disponíveis no Apêndice A.

6.1 Cadeira de Rodas

O robô usado para esse trabalho é uma cadeira de rodas motorizada modelo Jazzy 600.

Figura 6.1: Cadeira de rodas Jazzy 600.

Essa cadeira de rodas possui duas rodas padrão fixas motorizadas e quatro rodas Castor não-motorizadas. Como visto no Capítulo 2, as rodas Castor são omnidirecionais e não produzem restrições cinemáticas ao movimento da cadeira, e como não são motorizadas elas também não recebem sinais de controle. Do ponto de vista do modelo cinemático, portanto, elas podem ser desprezadas. Sua única função é garantir estabilidade estática ao sistema. O modelo dessa cadeira é o robô diferencial, dado por (2.25) e (2.26). A cadeira, portanto, é não-holonômica, pois não se desloca lateralmente. Contudo, para utilizar esse modelo, as entradas do sistema deveriam ser as velocidades angulares dos motores das rodas, o que não é o caso. As entradas são as velocidades linear e angular do ponto localizado no eixo de rotação da cadeira (entre as rodas padrão), que foi usado como configuração do robô e origem do sistema de coordenadas solidário ao seu corpo rígido. O eixo X_R está para frente, no sentido *do laser scanner*. A transformação da velocidade linear e angular da cadeira em velocidades angulares das rodas é executada pelo *driver* de potência da cadeira que envia os sinais para os motores. Dessa forma, embora seja um robô diferencial, a cadeira pode ser tratada como um uniciclo, cujo modelo é dado por (2.27). A posição do *laser scanner* acoplado é $[24cm\ 0\ 0]^T$ no sistema da cadeira. Esse ponto é a origem do sistema de coordenadas solidário ao *laser scanner*, que tem a mesma orientação que o sistema da cadeira.

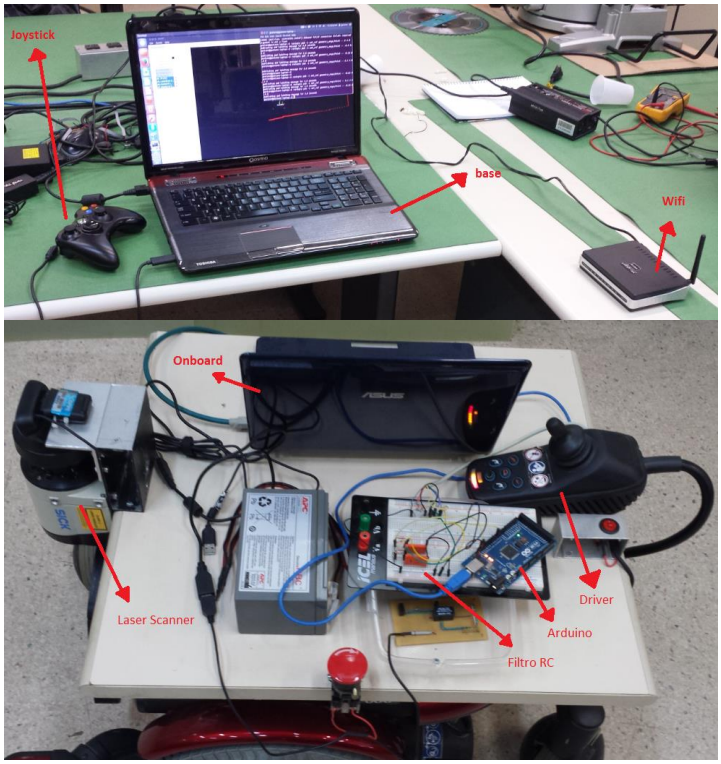


Figura 6.2: Montagem dos Componentes.

6.2 Montagem dos Componentes

A Figura 6.2 mostra a montagem dos componentes usados transmitir o sinal de velocidade à cadeira e a Figura 6.3 é o diagrama dessa montagem.

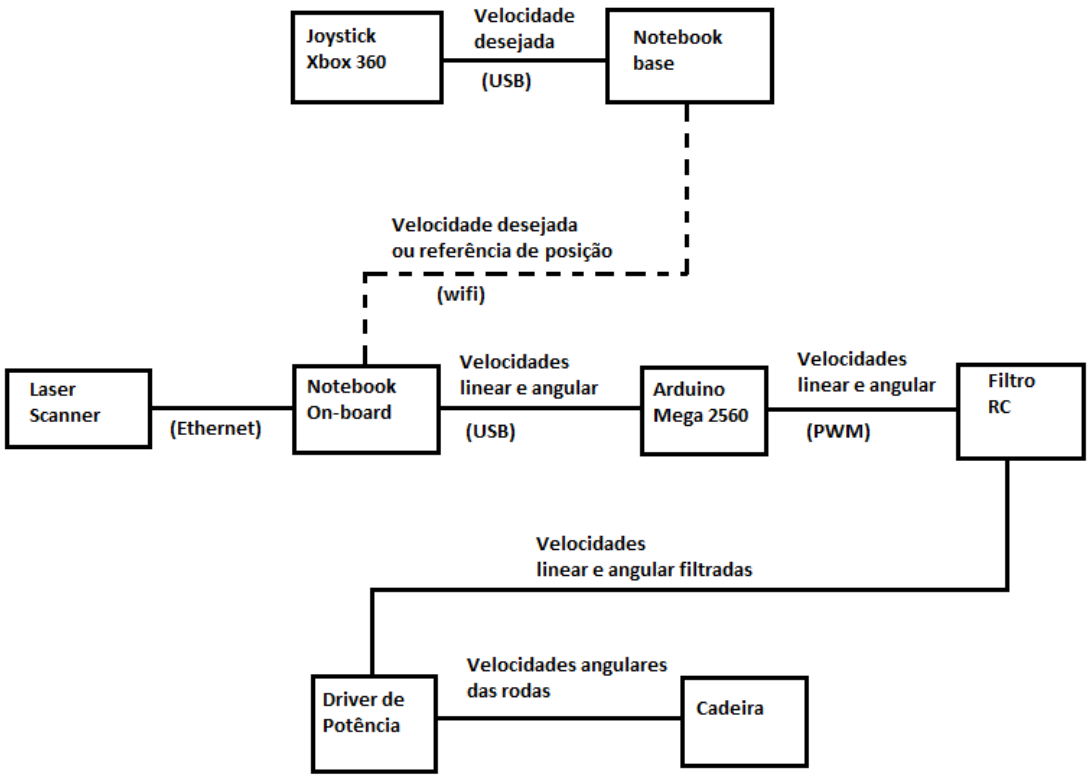


Figura 6.3: Diagrama de montagem dos componentes.

Para que a teleoperação seja possível, dois *notebooks* são conectados a uma rede local *wifi*. Um deles permanece em uma mesa (base) conectado ao *joystick* com o qual o usuário determina para onde a cadeira deve ir, determinando a velocidade desejada. O usuário também pode determinar uma referência de configuração diretamente por esse computador e monitorar as condições da cadeira remotamente através dele. O outro *notebook* é o computador de bordo que fica sobre a cadeira e é responsável por executar os programas que geram os sinais de velocidade que devem ser enviados à cadeira. Como o desvio de obstáculos depende das medições do ambiente, o computador de bordo também está conectado ao *laser scanner* pelo cabo *Ethernet*. Os programas são executados através do *software Robot Operating System* (ROS), que está instalado e é executado nos dois computadores. Como visto à frente, o ROS atua de forma distribuída

e trata os dois computadores como se fossem um, pois estão conectados na mesma rede. O sinal de velocidade gerado pelo computador de bordo está na forma de mensagem do ROS, que não é útil para o *driver* da cadeira. O sinal é enviado ao microcontrolador Arduino, que deve transformar a mensagem em sinal de tensão. Como o sinal de saída do Arduino é PWM e tem ruídos, foi colocado um circuito RC (filtro passa-baixas) para suavizar o sinal enviado ao *driver*, que transforma as velocidades linear e angular da cadeira em velocidades angulares das rodas e envia esses sinais de tensão aos motores. O *driver* da cadeira aceita tensões na faixa 1-4V, sendo 2,5V o ponto morto, acima de 2,5V é o sinal positivo de velocidade (para frente para velocidade linear e sentido anti-horário para velocidade angular) e menos de 2,5V para o sentido negativo de velocidade. Embora a maioria dos circuitos utilize a faixa de 0-5V, essa faixa de 1-4V é usada por questões de segurança, já que se trata de um equipamento para pessoas com deficiência física. Se um sinal nas faixas 0-1V ou 4-5V chega ao *driver*, ele se desarma automaticamente.

Nas próximas seções serão descritas as ferramentas utilizadas para o trabalho.

6.3 Microcontrolador Arduino

Microcontrolador é um pequeno computador em um único circuito integrado, contendo memória volátil e permanente, processador e periféricos programáveis de entrada e saída. É projetado para ser usado em aplicações que envolvam sistemas embarcados, como robôs, por exemplo. Entre suas principais características estão seu pequeno tamanho, baixo consumo de energia, baixo custo, capacidade de resposta em tempo-real às interrupções e normalmente ser dedicado a uma aplicação específica, o que permite a otimização de sua performance, qualidade e possíveis ganhos de escala devido à produção em massa, embora existam microcontroladores de uso geral.

Consequentemente, programas escritos para microcontroladores são pequenos, pois devem caber na sua pequena memória permanente. Para alguns microcontroladores, os programas devem ser escritos em linguagem criadas especificamente para eles, mas a maioria compreende linguagens de uso geral como C/C++ ou versões modificadas dessas linguagens. Eles também têm seus próprios compiladores para converter os códigos escritos em linguagem de alto nível para código de máquina. Microcontroladores normalmente também têm algumas dezenas de pinos de entrada e saída, tanto analógicos quanto digitais. Esses pinos podem ser configurados

por *software* para agirem como entradas para ler sinais de medida vindos de sensores externos, ou como saídas, controlando aparelhos externos, como lâmpadas de LED ou motores.

Outras funcionalidades do microcontrolador expandem as aplicações dos pinos. Conversores A/D podem ser usados para que pinos digitais sejam capazes de ler sinais vindos de sensores analógicos e conversores D/A permitem que os pinos digitais configurados como saídas enviem sinais analógicos para os aparelhos a serem controlados. Também é comum que alguns pinos digitais tenham a função PWM, que é muito eficaz para controlar motores, como visto adiante.

Muitas empresas fabricam placas de circuito impresso que contém microcontroladores embutidos, além de outros circuitos integrados e conectores de cabos (USB, *Ethernet*, entre outros), o que poupa o tempo de desenvolvedores que não desejam ter que montar os circuitos e facilita o aprendizado de iniciantes. Uma das placas mais comuns é o Arduino, que tem custo relativamente baixo e uma grande comunidade que desenvolve códigos para essa plataforma. Existem vários modelos de Arduino, normalmente variando o número de pinos e frequência de operação dos microcontroladores.

Os programas para Arduino são escritos em computadores comuns através de um *software* de interface (*interface development environment*, IDE) próprio que deve ser instalado no computador. Essa IDE é *cross-platform*, rodando em Windows, Linux e Macintosh OSX, e possui muitas funções que facilitam o desenvolvimento de aplicações mesmo para quem não é familiarizado com programação. A linguagem usada para escrever no Arduino é o Processing. Depois de escrito no computador, o programa é baixado para o Arduino por USB através da IDE.
Os programas de Arduino consistem de dois blocos: o bloco da função *setup()*, que é executado apenas uma vez quando o Arduino é inicializado e serve basicamente para configurar os pinos que serão usados como entradas ou saídas usando a função *pinMode()*; e o bloco da função *loop*(), que é executada continuamente até que o Arduino seja desligado ou que o botão de RESET seja pressionado. Nesse bloco são escritos os comandos que o Arduino deve executar. O modelo usado para esse trabalho é o Arduino Mega 2560.

6.3.1 PWM

O PWM (*Pulse-Width Modulation*) é um tipo de modulação de sinal que controla a largura de uma onda retangular. O valor médio de tensão e corrente que é entregue ao motor é controlado fechando e abrindo um *switch* entre a fonte e a carga com uma alta frequência. Quanto mais tempo um *switch* passa fechado em relação ao período da onda, mais potência é fornecida para a carga. Essa relação entre o tempo que o *switch* passa fechado e o período é chamada de *duty cicle*.

O valor médio de um sinal periódico *f(t)* com período T é dado por:

$$\overline{y} = \frac{1}{T}\int_0^T f(t)dt. \tag{6.1}$$

Para a onda retangular essa integral se torna:

$$\overline{y} = \frac{1}{T}\left(\int_0^{DT} y_{max}dt + \int_{DT}^{T} y_{min}dt\right) = \frac{DTy_{max}+T(1-D)y_{min}}{T}.$$

O valor médio da onda retangular é dado por:

$$\bar{y} = D.y_{max} + (1-D)y_{min}\ , \tag{6.2}$$

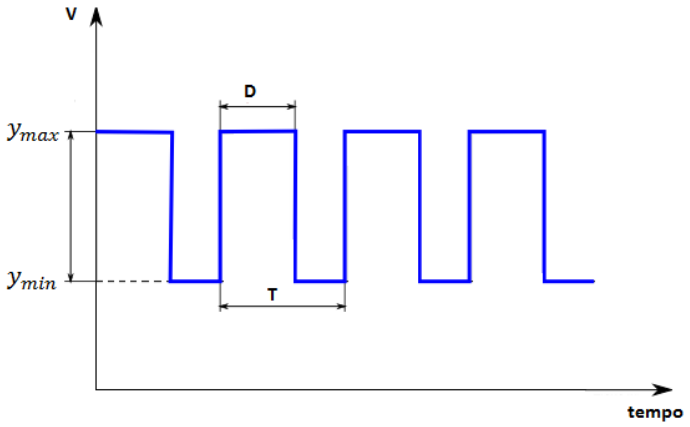


Figura 6.4: Sinal de tensão PWM.

sendo D o *duty cicle*, y_{max} o valor da tensão quando o *switch* está fechado e y_{min} o valor da tensão quando o *switch* está aberto. Normalmente, em aplicações práticas, $y_{min} = 0V$, e com isso, o valor médio é $\bar{y} = Dy_{max}$, proporcional ao *duty cicle.* Para o Arduino, $y_{max} = 5V$.

Quando o *switch* está aberto, pouca corrente passa, e quando está fechado, há pouca diferença de potencial, e por isso a dissipação de potência do PWM é muito baixa. Dessa forma, o PWM é um modo de transmitir potência para o motor de forma muito mais eficiente energeticamente do que um potenciômetro, que dissipa muita energia em forma de calor.

6.4 *Laser Scanner*

Um *Laser Scanner*, ou LIDAR, é um aparelho que lança um raio *laser* contra um objeto, que é refletido de volta para determinar a distância entre o próprio aparelho e o objeto. Um receptor detecta a componente do *laser* de retorno que é coaxial ao *laser* transmitido (parte da luz é difratada ao atingir o objeto). O raio é lançado somente em uma direção a cada instante, mas um sistema mecânico com um espelho que se move rapidamente direciona o raio em diversos ângulos, permitindo que o o aparelho cubra um cenário inteiro em 2D ou 3D em pouco tempo. É um dos sensores mais populares em robótica móvel para detecção de obstáculos por ter custo relativamente baixo e medidas de fácil interpretação que podem ser usadas diretamente em uma aplicação.

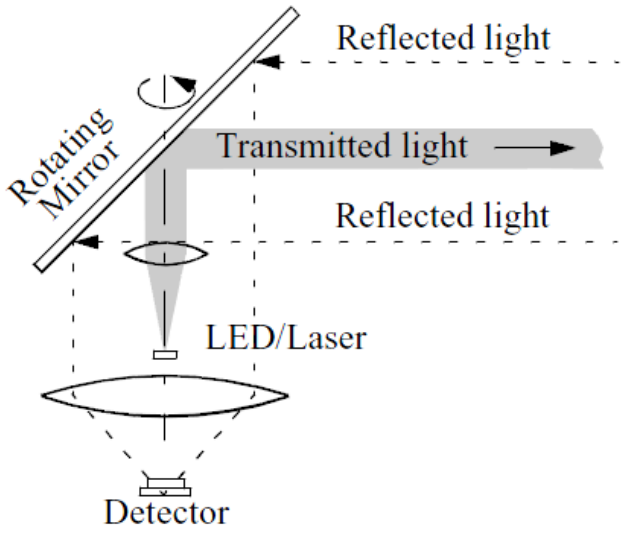


Figura 6.5: Esquemático de um *laser scanner* [4].

O modo mais comum de determinar a distância a partir desse raio é chamado *time-of-flight*, que consiste em medir o tempo que o raio leva para ir até o objeto e voltar até o aparelho. Esse tempo é medido através da diferença de fase entre a luz transmitida e a recebida por um opto-receptor conectado a um circuito eletrônico do *laser scanner*.

Como a velocidade da luz no meio é conhecida, medir a distância é simples: Seja *D* a distância entre o aparelho e o objeto, *c* a velocidade da luz no meio, *t* o intervalo de tempo que a luz leva para ir e voltar ao aparelho, φ o atraso de fase da luz medida pelo aparelho e *f* a frequência da luz, que é conhecida:

$$D = \frac{ct}{2} \quad , \quad t = \frac{\varphi}{2\pi f},$$

e substituindo temos:

$$D = \frac{c\varphi}{4\pi f}. \tag{6.3}$$

Para tornar a medida mais precisa, vários raios são lançados consecutivamente na mesma direção e a distância medida é média de todos os lançamentos.

A qualidade da medida do *laser* depende da superfície que ele atinge. Para superfícies opacas, as difrações são mínimas e o laser é extremamente preciso. Para superficies espelhadas, como vidro e metais polidos, a incerteza é tão alta que o *laser* se torna inútil.

A incerteza da distância depende da amplitude do sinal recebido, portanto existe uma distância máxima para que a medida do sensor seja confiável. A luminosidade do lugar tambem afeta a medida. Quanto mais claro, maior a amplitude do sinal e menor a incerteza. O uso desse sensor depende, portanto, do ambiente em que o robô irá trabalhar.

Os dados do *laser scanner* são transmitidos como um *array* que contém as coordenadas dos pontos obtidos, normalmente em coordenadas polares, que podem ser visualizados em programas específicos como o *rviz*, para ROS, como visto na Figura 6.6.

O *laser scanner* usado para esse trabalho é um modelo SICK LMS111-10100 que capta pontos em uma abertura de 270^o ($0{,}5^o$ entre dois pontos, totalizando 541 pontos)

a uma distância máxima de 20 metros, portanto ele não enxerga 90^o para trás. A frequência do *laser* é 25 Hz.

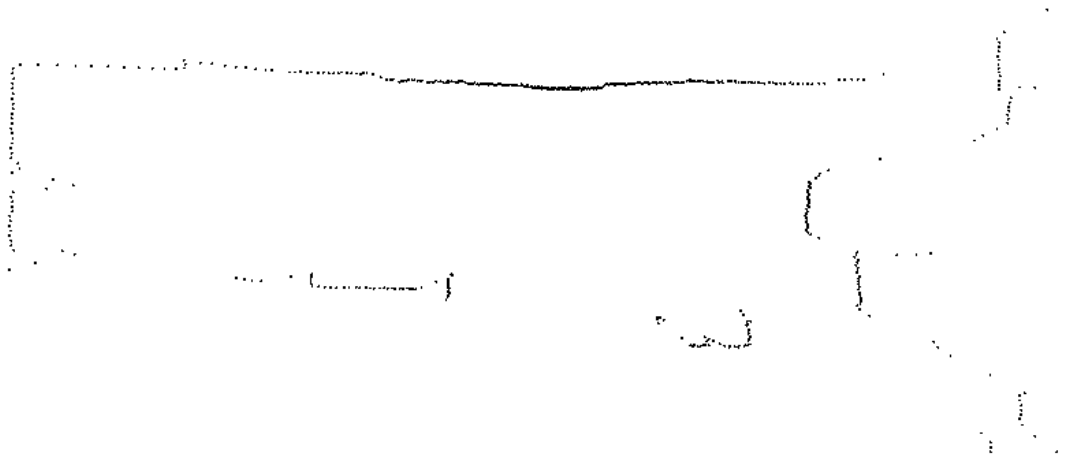


Figura 6.6: Visualização do conjunto de pontos obtido pelo *laser scanner* no *rviz.*

6.5 Eletroencefalografia (EEG):

O EEG é a gravação de atividade elétrica do cérebro. Mede flutuações de tensão resultantes das correntes elétricas entre os neurônios através de múltiplos eletrodos colocados ao longo do escalpo com um gel condutor para facilitar a leitura dos sinais. Os sinais obtidos pelos eletrodos são processados por filtros digitais para atenuação de ruídos.

Para o EEG usado nesse trabalho foram usados 8 eletrodos conectados ao escalpo do operador. O operador permanece diante de um monitor de computador com 4 lâmpadas nas bordas: uma em cima, uma embaixo, uma à esquerda e uma à direita. Cada lâmpada pisca em uma frequência diferente e quando o operador deseja dar um comando de direção à cadeira, ele deve olhar fixamente para a lâmpada correspondente. Os sinais cerebrais são estimulados na frequência da lâmpada e podem ser lidos no EEG. Um programa decide o sentido em que a cadeira deve se mover e envia essa mensagem por *Bluetooth* para o Arduino que está sobre a cadeira.

6.6 ROS

O *Robot Operating System* (ROS) é um *framework* flexível e *open source* usado para aplicações em robótica. É uma coleção de ferramentas, bibliotecas e convenções que tem o objetivo de simplificar a tarefa de criar robôs que executem funções das mais simples às mais complexas de forma robusta em uma grande variedade de plataformas.
Devido à sua natureza multidisciplinar, criar *software* de propósito geral em robótica é uma tarefa complicada que exige conhecimento em diversas áreas. Partindo dessa constatação, o ROS foi projetado para encorajar o desenvolvimento cooperativo. Como exemplo, pesquisadores que trabalham na área de mapeamento podem se beneficiar de aplicações desenvolvidas por outros grupos que trabalham nessa área e também podem contribuir para que outros pesquisadores especializados em outras áreas, como navegação, usem suas aplicações de mapeamento, no qual eles teoricamente tem menos conhecimento que o grupo anterior. Dessa forma, se alguém decide iniciar um projeto, não é necessário começar do zero e aprender de forma aprofundada sobre todas as demais áreas relacionadas ao assunto do projeto. Cada um pode construir seu projeto a partir de outros que já existam (códigos reutilizáveis).

Apesar do nome, o ROS não é um sistema operacional e sim um *middleware*. O *middleware* é um programa que providencia serviços para aplicações que não existem em sistemas operacionais (OS) convencionais. Também possibilita a comunicação e gerenciamento de dados entre *softwares* desenvolvidos para OS diferentes agindo como uma camada entre eles. É especialmente útil para sistemas distribuídos, permitindo que cada máquina do sistema com seu próprio OS e arquitetura de *hardware* possa executar funcionalidades comuns de OS através de protocolos comuns, como se fossem todas iguais. Entre essas funcionalidades estão abstração de *hardware*, controle de periféricos, troca de mensagens e acesso a serviços e processos de outras máquinas.
Os softwares de ROS se dividem em três categorias:

- Ferramentas independentes de plataforma e linguagem para distribuição e *building* (compilação e ligação) dos demais *softwares*: *rosbuild* e *catkin*;
- Bibliotecas clientes, que são as implementações do ROS em diversas linguagens, como *roscpp* (C++), *rospy* (Python) e *roslisp* (LISP);
- Pacotes que contêm códigos relacionados a aplicações e que usam uma ou mais bibliotecas clientes.

Embora sejam *cross-platform,* atualmente as bibliotecas clientes do ROS funcionam somente em sistemas baseados em Unix (Linux e Mac) e são mais estáveis em Ubuntu. Por esse motivo, o OS usado nos *notebooks* para esse trabalho é o Ubuntu Precise. A versão usada do ROS no *notebook* de bordo é o Hydro e no *notebook* base é o Fuerte, uma versão mais antiga. Todos os códigos necessários para que a cadeira se desloque são executados no *notebook* de bordo e como a base é usada somente para monitoração, não há nenhum problema de incompatibilidade entre as duas versões.

6.6.1 Conceitos:

Pacote: *Software* em ROS é organizado em estruturas chamadas pacotes, que basicamente são diretórios que contêm todos os arquivos necessários para que uma aplicação seja executada. Um pacote pode conter nós, bibliotecas independentes do ROS que sejam necessárias para o seu funcionamento, conjuntos de dados e arquivos de configuração. O pacote é a unidade atômica do ROS, ou seja, a menor unidade que pode ser compilada e é a forma como toda aplicação é distribuída. Para que um diretório seja visto como um pacote pela biblioteca cliente, ele deve ter dois arquivos:

- *package.xml* (manifesto do pacote) define propriedades do pacote como o seu nome, versão, nome do autor e suas dependências de outros pacotes;
- *CMakeList.txt* é o arquivo de entrada de dados para o CMake (sistema de *building*). Nesse arquivo devem estar descritos todos os nós e bibliotecas que pertencem ao pacote.

O comando usado para iniciar o *building* de todos pacotes é *catkin_make*. Esse comando deve ser digitado em um terminal, no diretório do *workspace* do ROS, o diretório onde todos os pacotes devem estar.

Nó: É o nome que o ROS usa para designar um processo, ou seja, um código executável com uma ou mais funções que é executado continuamente até ser encerrado. Todo nó pertence a um pacote e toda aplicação feita em ROS é baseada em um conjunto de nós que podem pertencer a diferentes pacotes. Para esse trabalho, por exemplo, há nós que agem como *drivers* para os periféricos usados (*laser scanner*, *joystick* e Arduino), outros que implementam os algoritmos nesse trabalho, como o APF, o PlICP e o

RANSAC. Não podem existir dois nós com o mesmo nome em um mesmo pacote. Nós podem ser escritos em qualquer linguagem para a qual exista uma biblioteca cliente, embora a maioria deles seja escrita em C++ ou Python (usando funções de *roscpp* ou *rospy*, respectivamente). Nós escritos em linguagens diferentes se comunicam normalmente.

Um nó é executado digitando o seguinte comando em um terminal:

rosrun nome_do_pacote nome_do_nó

O pacote *rosnode* possui ferramentas de linha de código que ajudam a monitorar o que acontece com os nós. Os principais comandos são:

-Para saber quais nós estão sendo executados em um dado momento, basta digitar:

rosnode list

-Para verificar os tópicos para os quais o nó se inscreve e publica, o comando é:

rosnode info nome_do_nó

Mensagem: Nós se comunicam publicando mensagens em tópicos. Uma mensagem é uma estrutura de dados que contém um ou mais campos, que podem ser tipos primitivos (*int*, *float*, *bool*, etc) ou mensagens de outros tipos, além de *arrays*. A mensagem, portanto, é o mecanismo de entrada e saída de argumentos necessários para que um nó execute as funções contidas nele. Toda mensagem tem um tipo que indica quantos campos ela possui e quais os tipos desses campos. Por exemplo, mensagens do tipo *geometry_msgs/Point* são formadas por três membros: *float x, float y e float z,* que representam as coordenadas de um ponto no R^3. Todo tipo de mensagem também pertence a um pacote, cujo nome é o termo que aparece antes da barra. O tipo usado no exemplo acima pertence ao pacote *geometry_msgs*. Um usuário pode criar novos tipos de mensagem em seus próprios pacotes.

Tópico: É o sistema de comunicação do ROS, através do qual as mensagens são trocadas pelos nós. Não podem existir dois tópicos com o mesmo nome em um dado momento. Se um nó se inscreve para um tópico, isso significa que ele está constantemente verificando se uma nova mensagem chega nesse tópico. Assim que a mensagem chega, o nó imediatamente recebe a mensagem. Quando um nó quer transmitir uma mensagem de um certo tipo, ele publica essa mensagem em um tópico, especificando seu nome e o tipo de mensagem. Todos os nós que estiverem inscritos nesse mesmo tópico receberão essa mensagem. A troca de mensagens por tópicos permite a comunicação anônima entre nós em um sistema distribuído, desacoplando a produção e o consumo de informação, pois quando um nó recebe uma mensagem em um tópico, ele não sabe que outro nó gerou essa mensagem. Da mesma forma, quando um nó publica uma mensagem em um tópico ele não sabe que nós estão inscritos nesse tópico e terão acesso à mensagem. Cada nó pode publicar e se inscrever em qualquer quantidade de tópicos. Cada tópico pode aceitar somente um tipo de mensagem.

O pacote *rostopic* possui ferramentas de linha de código que ajudam a monitorar o que acontece nos tópicos. Os comandos mais importantes são:

-Para ver todos os tópicos utilizados em um dado momento, basta digitar o seguinte comando no terminal:

rostopic list

-Para ver o tipo de mensagem de um certo tópico e quais nós se inscrevem e publicam nele, o comando é:

rostopic info nome_do_tópico

-Para ver as mensagens que estão sendo publicadas no tópico, o comando é:

rostopic echo nome_do_tópico

-O usuário também pode publicar diretamente uma mensagem no tópico, usando o comando:

rostopic pub -1 nome_do_tópico tipo_de_mensagem -- [mensagem]

Esse comando pode ser usado no programa APF que foi criado para esse trabalho para escolher a referência da cadeira de rodas, como visto adiante. O tópico é */set_ref* e o tipo de mensagem é *geometry_msgs/Point*.

-Para verificar a frequência com que mensagens são publicadas no tópico, o comando é:

rostopic hz nome_do_tópico

Mestre: É o serviço de registro e nomeação do ROS. Ele indica para os nós os endereços dos tópicos para os quais eles querem se inscrever ou publicar. O objetivo do Mestre é apenas auxiliar os nós a localizarem os tópicos no momento em que eles são criados. Depois de localizados, a comunicação entre nós e tópicos é direta, sem influência do Mestre. Uma sessão de ROS é iniciada quando o Mestre é iniciado, o que ocorre quando o comando *roscore* é digitado em um terminal.

6.6.2 Estrutura de um nó

Todos os nós para esse trabalho foram escritos em C++, logo as linhas de código abaixo não devem ser usadas em nós escritos em outras linguagens, mas em toda linguagem há comandos com os mesmos objetivos.

O código começa incluindo todos os cabeçalhos necessários para o seu funcionamento, tanto os que contém os tipos de mensagem utilizadas quanto os de bibliotecas externas. Os tipos de mensagem são implementados como classes.

Em seguida são escritas as definições das funções *Callback* que recebem as mensagens *msg* como argumentos. Funções *Callback* são funções executadas apenas quando uma interrupção ocorre, nesse caso a interrupção é a chegada de uma mensagem ao nó. A definição de uma função *Callback* tem a forma:

```
void CallBackFunction(const tipo_de_mensagem::ConstPtr& msg)
{
        …codigo…
}
```

Cada função *Callback* recebe apenas mensagens de um tipo como argumento. Depois das definições *Callback* temos o bloco principal, contido na função *main():*

```
int main(int argc, char **argv)
{
        ros::init(argc, argv, "nome_do_no");
        ros::NodeHandle n;
        ros::Publisher pub = n.advertise<tipo_da_mensagem>("nome_do_top
ico", tamanho_da_fila);
        ros::Subscriber sub = n.subscribe("nome_do_topico", tamanho_da_
fila, CallbackFunction);

        ros::Rate loop_rate(10);
        while (ros::ok())
        {
                spinOnce();

                …codigo…

                loop_rate.sleep();
        }
}
```

ros::init() deve ser sempre a primeira função chamada por *main()*. É a função que inicia o ROS e nomeia o nó. Cada nó deve ter pelo menos um objeto *NodeHandle*. Quando o primeiro *NodeHandle* é criado, o nó é criado e quando o último *NodeHandle* é destruído, o nó é destruído. A função *advertise()* indica que o nó quer publicar para um tópico. Entre < > está o tipo da mensagem e entre ( ) está o nome do tópico e o tamanho máximo da fila de mensagens consecutivas que o tópico guarda caso nenhum outro nó esteja recebendo as mensagens. A função *advertise()* deve ser chamada para cada tópico que o nó quiser publicar. A função *subscribe()* indica que o nó quer se inscrever para um tópico. Entre ( ) estão o nome do tópico, o tamanho máximo da fila de mensagens recebidas que o nó guarda e o nome da função *Callback* que usará essa mensagem como argumento. A função *subscribe()* deve ser chamada para cada tópico para o qual o nó quiser se inscrever.

Depois disso começa a execução cíclica do nó. O objeto *loop_rate* e seu método *sleep()* não são obrigatórios, mas devemos utilizá-los para que a frequência de execução

do nó seja constante. A função *ros::ok()* retorna um booleano: *true* se nada acontecer e *false* caso haja uma ordem para que o nó pare, por exemplo, digitar o comando Ctrl-C no terminal para matar o nó.

A função *spinOnce()* permite que as funções *Callback* sejam executadas. Quando o nó recebe uma mensagem, ele não executa a *Callback* imediatamente. Ao invés disso, as mensagens são acumuladas em uma fila. Apenas quando *spinOnce()* é executada, o nó verifica a fila e passa as mensagens para as suas respectivas *Callbacks* até que a fila se esvazie. Mensagens que cheguem após a execução de *spinOnce()* tem que esperar até a próxima execução dessa função.

6.6.3 Implementação dos Algoritmos:

Podemos agora descrever os pacotes usados para esse projeto. Alguns foram baixados da *internet* e outros foram criados pelo autor especificamente para esse trabalho.

Pacotes baixados da *internet*:

Joy: Esse pacote possui o nó *joy_node*, que age como *driver* para *joysticks* genéricos de Linux e Xbox 360 se comunicarem com o ROS. Um *driver* de dispositivo é um programa que serve de interface entre os sinais elétricos gerados pelo *hardware* de um dispositivo e um *software* instalado em um computador ao qual o dispositivo está conectado, traduzindo os sinais elétricos para um tipo de dados que o *software* compreenda. Os comandos do *joystick* são tratados pelo ROS como mensagens do tipo *sensor_msgs/Joy* publicadas pelo tópico */joy*.

lms1xx: Esse pacote possui o nó *LMS1xx_node*, que age como *driver* para os *laser scanners* da série lms1xx da empresa SICK se comunicarem com o ROS. Os conjuntos de pontos gerados pelo *laser scanner* são tratados pelo ROS como mensagens do tipo *sensor_msgs/LaserScan* e publicados pelo tópico */scan*.

rosserial_python e rosserial_arduino: Esses pacotes criam um protocolo de comunicação serial entre ROS e o Arduino, permitindo que o Arduino aja como um nó de ROS, podendo publicar e se inscrever em tópicos para troca de mensagens. Rosserial_python cria o protocolo do lado servidor (o *notebook* de bordo) e rosserial_arduino cria o protocolo do lado cliente (o Arduino). O Arduino é representado no ROS pelo nó *serial_node.py*.

TF: Esse pacote acompanha as relações entre os diversos sistemas de coordenadas usados em uma aplicação ao longo do tempo, calculando as transformações homogêneas entre eles e permitindo a transformação de coordenadas de entidades (pontos, retas, etc) entre esses sistemas. Nesse trabalho são usados três sistemas de coordenadas: */world* (sistema inercial), */base_link* (sistema do corpo rígido da cadeira de rodas) *e /laser* (sistema do *laser scanner*). As mensagens são do tipo *tf2_msgs/TFMessage* e publicadas no tópico */tf*.

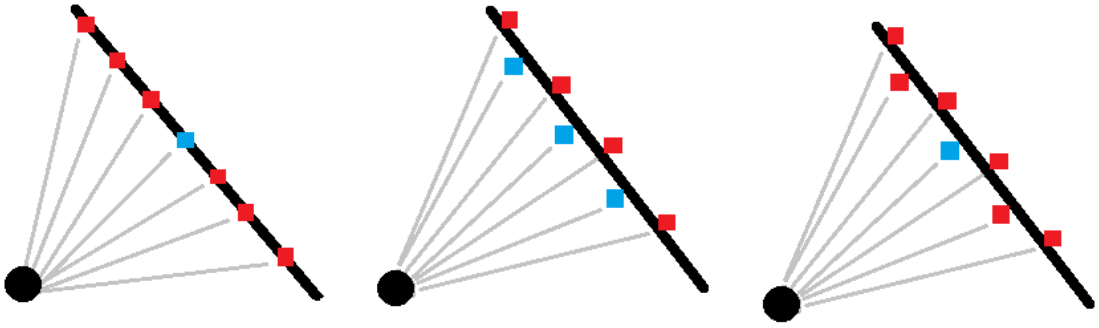
laser_scan_matcher: Esse pacote implementa o algoritmo PlICP de localização descrito no Capítulo 4. O nó *laser_scan_matcher_node* se inscreve para o tópico */scan* para receber mensagens do *laser scanner* e publica a posição do robô no sistema de coordenadas */world* como uma mensagem do tipo *geometry_msgs/Pose2D* no tópico */pose2D*. A posição do robô também é dada como uma transformação TF de */world* para */base_link* no tópico */tf*.

PointCloud Library (PCL): O PCL é uma biblioteca *open-source* originalmente desenvolvida como um pacote de ROS, mas que se tornou independente. Possui inúmeros algoritmos para processamento de nuvens de pontos e imagens em 2D e 3D em diversas áreas como filtragem, extração de características, reconhecimento de objetos e reconstrução de superfícies. O pacote de ROS agora age como uma integração entre os dois *softwares*. Para esse trabalho, apenas a implementação do PCL para o RANSAC foi usada.

Pacotes desenvolvidos pelo autor:

obstacle_avoidance: Esse pacote possui um único nó, *avoidance_teleop*, que se inscreve para os tópicos */joy* e */scan*. O objetivo é controlar a cadeira manualmente com o *joystick* e o nó simplesmente evita que ela atinja obstáculos. Não há referência para a cadeira seguir. Por isso, apenas o campo repulsivo do APF está implementado. O nó publica mensagens do tipo *geometry_msgs/Point* no tópico */veloc_arduino*. Essa mensagem é o sinal de velocidade linear e angular resultante da soma da velocidade desejada dada pelo *joystick* com os sinais repulsivos dos obstáculos sobre os vértices da cadeira. Como visto no Capítulo 3, a única variável desconhecida para calcular o sinal repulsivo é a distância de cada vértice até seus obstáculos mais próximos. Para determinar quais pontos do *laser* são os obstáculos mais próximos de cada vértice, basta verificar quais tem a distância menor que a dos seus vizinhos à esquerda e à direita, ou

seja, os mínimos locais. Na prática, devido à incerteza do *laser*, muitos pontos próximos seriam mínimos e a resultante dos seus sinais repulsivos seria desnecessariamente grande. Para resolver esse problema, basta que, ao invés de verificar apenas um vizinho de cada lado, sejam verificados *n_min* vizinhos de cada lado, sendo *n_min* um parâmetro do nó escolhido pelo usuário. Dessa forma, *n_min* age como um limitador na quantidade de mínimos locais próximos. Quanto maior *n_min*, maior a distância mínima entre dois mínimos locais. A Figura 6.7 ilustra esse processo:



a) b) c)

Figura 6.7: Mínimos locais de um *laser*

Em a) o *laser* idealmente não tem incertezas, colocando todos os pontos perfeitamente sobre a reta. Para verificar se um ponto é mínimo local, basta comparar sua distância com a de um vizinho de cada lado. Em b) o *laser* tem incertezas e encontra muitos mínimos locais próximos. Em c) com *n_min=3*, o *laser* tem apenas um mínimo local.

Os outros parâmetros do nó são *q_max* e *e*, sendo *q_max* o raio de influência do campo repulsivo e *e* a escala do sinal repulsivo, como em (3.10). As forças *f* nos vértices e *u* na origem do sistema de coordenadas do robô estão relacionadas por (3.15).

Essa equação está no sistema fixo e por isso é necessário encontrar o valor de θ, a orientação do sistema do robô em relação ao sistema inercial. Colocando a equação no sistema de coordenadas do robô temos:

$$\begin{bmatrix} u_x \\ u_y \\ u_\theta \end{bmatrix} = \begin{bmatrix} f_x \\ f_y \\ -f_x a_y + f_y a_x \end{bmatrix}. \tag{6.4}$$

Com essa equação, não é necessário obter θ e, portanto, não precisamos da localização do robô para essa aplicação. Por esse motivo, o nó *avoidance_teleop* está escrito no sistema do robô. Além disso, como a cadeira não se move lateralmente, não pode haver sinal de velocidade lateral, portanto $u_y = 0$ sempre.

apf: Esse pacote possui um único nó, *apf*, que se inscreve para os tópicos */joy*, */scan* e */set_ref*. Esse pacote implementa o planejamento de trajetória usando o APF completo, com o potencial atrativo de uma referência. A referência é uma mensagem do tipo *geometry_msgs/Point* publicada em */set_ref*. Essa referência é dada pelo usuário digitando no terminal:

rostopic pub -1 set_ref geometry_msgs/Point -- x y z

A mensagem representa as coordenadas *x* e *y* da referência no sistema de coordenadas inercial, em metros, e *z* é a orientação, em radianos. Esse nó também publica mensagens do tipo *geometry_msgs/Point* no tópico */veloc_arduino*. Essas mensagens são o sinal de velocidade resultante dos sinais de velocidade do *joystick* (implementado por segurança), e dos sinais atrativo e repulsivo. O sinal repulsivo é calculado da mesma forma que no pacote *obstacle_avoidance*. O sinal atrativo precisa da localização do robô e, portanto, o nó do pacote *laser_scan_matcher* precisa ser executado em conjunto com esse. Apenas os dois vértices da frente sofrem influência do campo atrativo, já que apenas dois pontos são necessários para determinar um corpo rígido.

avoidance_bio: Esse pacote é igual ao *obstacle_avoidance*, implementando apenas os campos repulsivos dos obstáculos. A diferença é que ao invés dos comandos de velocidade serem dados pelo *joystick*, eles são dados pelas ondas cerebrais captadas pelo EEG. O sentido em que a cadeira deve se deslocar é dado por 8 bits, 2 bits para cada sentido (frente, trás, esquerda e direita). A cada instante, o comando pode ser dado em apenas um sentido, não é possível enviar sinal para que a cadeira se mova para frente e para a esquerda ao mesmo tempo, por exemplo. Os 8 bits são transmitidos para o Arduino através de uma placa Bluetooth e publicados no tópico */dataReading* como mensagens do tipo *std_msgs/UInt8* pelo nó do Arduino. O nó *avoidance_bio* se inscreve para esse tópico e determina a velocidade desejada. Com 2 bits para cada sentido existem 4 módulos possíveis para a velocidade, que são números entre 0 e 1, assim como no *joystick* (entre 0 e 1 para frente e esquerda e entre -1 e 0 para trás e direita). A

partir de então o algoritmo é o mesmo que para o nó *avoidance_teleop* do pacote *obstacle_avoidance*.

line_recognition: Esse pacote permite que a cadeira identifique os corredores pelos quais está se deslocando. Se uma pessoa com deficiência física estiver usando a cadeira para se deslocar pelos corredores, ela não poderá usar comandos complexos como os do *joystick* analógico. São necessários comandos simples como ir para frente, parar e virar no próximo corredor à esquerda ou à direita que possam ser dados por ondas cerebrais. Para esse pacote, os botões do controle do Xbox 360 são usados para dar esses comandos. Existem dois nós nesse pacote:

- *laser_to_pointcloud2*: esse nó apenas converte as mensagens do *laser scanner* obtidas pelo tópico */scan* para mensagens do tipo *sensor_msgs/PointCloud2* publicadas no tópico */pc2*. Isso é necessário porque a biblioteca PCL usada no nó abaixo trabalha apenas com esse tipo de mensagem;
- *line_rec*: esse é o nó principal. Ele se inscreve para os tópicos */pc2* e */joy* e tem como saída a referência que é enviada para o nó *apf* do pacote *apf*, que sempre deve ser executado em conjunto com esse. A referência é publicada no tópico */set_ref*, o mesmo para o qual *apf* se inscreve. Esse nó utiliza a implementação do algoritmo RANSAC que existe na biblioteca PCL para encontrar os parâmetros de todas as retas obtidas pelo *laser scanner* no sistema de coordenadas */laser*. Depois ocorre o processamento das retas, no qual retas muito parecidas se tornam uma só. O nó, então, verifica quais dessas retas estão próximas da cadeira seguindo o critério abaixo:

Para cada reta, é calculado o ponto de interseção entre ela e a reta perpendicular a ela que passa pela origem do sistema de coordenadas (o próprio *laser scanner*). Se pelo menos um *inlier* dessa reta estiver a uma distância menor que *d_int* (um parâmetro do nó escolhido pelo usuário) da interseção e a distância da própria interseção até a origem do sistema for menor que *dist_max* (outro parâmetro) essa reta é considerada próxima da cadeira. A Figura 6.8 ilustra esse método.

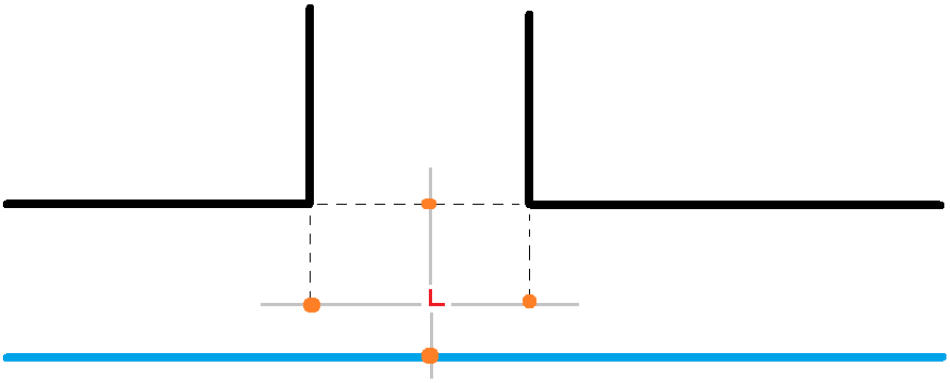


Figura 6.8: Determinação das retas próximas ao robô.

As retas pretas são retas encontradas pelo RANSAC que não estão próximas do robô. A reta azul está próxima do robô. As retas cinzentas são as perpendiculares e os pontos laranja são as interseções de cada reta encontrada pelo RANSAC com sua respectiva reta perpendicular. Com base nas retas próximas e no comando de direção do controle, a referência é atualizada a cada instante como mostrado nas situações da Figura 6.9, em que novamente as retas azuis são próximas e as referências são os pontos verdes:

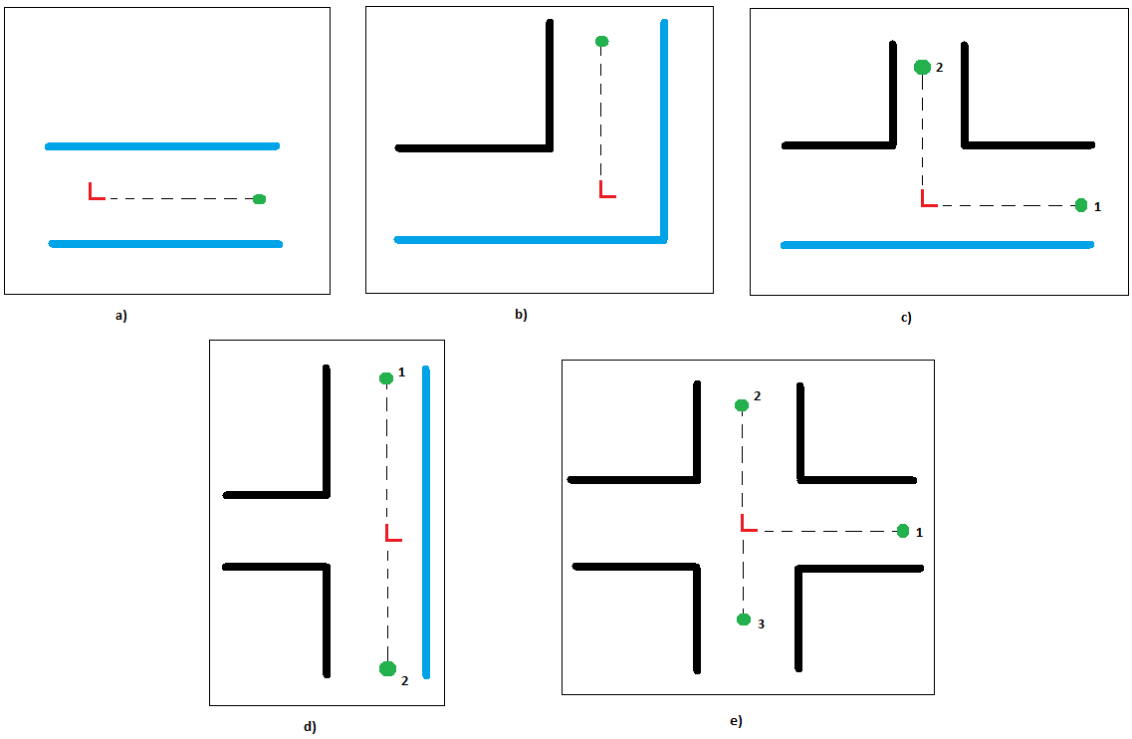


Figura 6.9: Configurações dos corredores.

Em a) existem duas retas próximas paralelas, o que significa que a cadeira está em um corredor e pode andar apenas para frente, mesmo que o comando seja para virar. A referência é colocada sobre uma reta com coeficiente angular igual à média dos coeficientes angulares das retas do corredor partindo da origem (a distância é um parâmetro).

Em b) existem duas paredes próximas perpendiculares entre si, o que significa que a cadeira está em uma curva em L, para um único lado. Para saber qual lado, basta verificar o coeficiente linear da parede horizontal. Se for negativo, a curva é para a esquerda e se for positivo, para a direita. Apenas o comando para virar para o lado certo é permitido, colocando a referência em uma reta paralela à parede próxima vertical. Qualquer outro comando faz a cadeira parar.

Em c) e d) existe uma parede próxima, o que significa uma curva em T.

Em c) a parede está na horizontal e é possível dar o comando para frente, colocando a referência em uma reta paralela à parede (referência 1) ou para virar para um dos lados. Mais uma vez o lado certo pode ser verificado através do coeficiente linear da parede: se negativo, a curva é para a esquerda e se positivo é para a direita. Nesse caso a referência é colocada sobre uma reta perpendicular à parede (referência 2). Se for dado o comando para virar para o lado errado, a cadeira segue em frente.

Em d) a parede está na vertical e só é possível virar para os dois lados. A referência fica sobre uma reta paralela à parede e os sinais das suas coordenadas dependem do lado escolhido, esquerda (referência 1) ou direita (referência 2). O comando para frente faz a cadeira parar.

Em e) não há retas próximas, o que significa um cruzamento. Todos os comandos são possíveis. Seguir em frente põe a referência no eixo $+X$ (referência 1), virar a esquerda põe no eixo $+Y$ (referência 2) e a direita põe no eixo $-Y$ (referência 3).

Por último, o comando de parar sempre põe a referência na configuração atual do robô.

Além desses pacotes, há também os programas para o Arduino:

wheelchair.ino: Usado quando se pretende usar o controle manual por *joystick* ou o planejamento de trajetória completo. Inscreve-se para o tópico */veloc_arduino* e converte os sinais de velocidade recebidos em sinais de tensão PWM que são enviados ao circuito RC.

wheelchair_bio.ino: Usado quando se pretende usar o controle por ondas cerebrais. Também se inscreve para o tópico */veloc_arduino* e converte os sinais de velocidade recebidos em sinais de tensão PWM, mas também recebe os comandos de direção do EEG por Bluetooth e os publica como mensagens do tipo *std_msgs/UInt8* no tópico */dataReading*.

6.7 Resultados:

Com os códigos explicados, podemos finalmente apresentar os resultados dos testes práticos. O computador de bordo responsável por executar os programas é um *netbook* Asus com processador Intel Core Duo e 2GB de memória RAM.

Os gráficos mostrados abaixo foram criados com o pacote *rqt_plot* do ROS, que os gerou a partir das mensagens publicadas em tópicos. Da mesma forma, os diagramas mostrando as conexões entre os nós necessários para executar as aplicações foram criados com o pacote *rqt_graph*. Os nomes em elipses são nós e em retângulos são tópicos. As imagens abaixo foram criadas pelo pacote de visualização *rviz*. Nas visualizações, os pontos vermelhos são obstáculos identificados pelo *laser scanner*, os quatro pontos cinzentos são os vértices da cadeira, tratada como um retângulo, os círculos com centro em cada vértice são as zonas de influência dos campos repulsivos que afetam os respectivos vértices, os pontos amarelos são a referência e a posição dos dois vértices frontais que correspondem à referência. Também estão presentes os sistemas de coordenadas usados.

Todos os nós necessários para cada aplicação estão contidos em um arquivo *launch*. O *launch* é um arquivo de configuração que permite iniciar vários nós simultaneamente, enquanto *rosrun* inicia apenas um. O arquivo *launch* é apenas um arquivo XML com a extensão *.launch*. Pertence a um pacote e para executá-lo basta digitar o seguinte comando no terminal:

roslaunch nome_do_pacote nome_do_arquivo.launch

Para todos os gráficos de sinais de velocidade, veloc_arduino/x é a velocidade linear e veloc_arduino/y é a velocidade angular.

6.7.1 Desvio de obstáculos:

Arquivo *Launch*: *run_avoidance.launch*

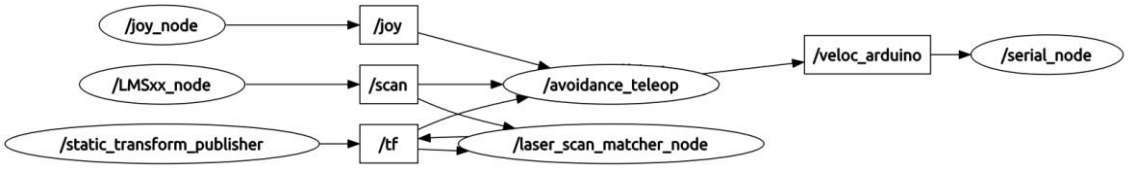


Figura 6.10: Diagrama de nós para o desvio de obstáculos.

Na situação abaixo (Figura 6.11), a cadeira está de frente para uma parede perpendicular a ela. O comando de velocidade para frente é dado em 14s e a cadeira anda em direção à parede até parar completamente em 15s, pois o sinal repulsivo passa a ter módulo igual ao sinal do joystick. Em 16.7s o comando de velocidade cessa e a cadeira anda de ré devido ao sinal repulsivo, até parar em 19,3s. O sinal de velocidade angular varia com alta frequência e uma média próxima de zero devido ao ruído do *laser*, e ao ser filtrado pelo circuito RC não afeta o movimento da cadeira, que é reto.

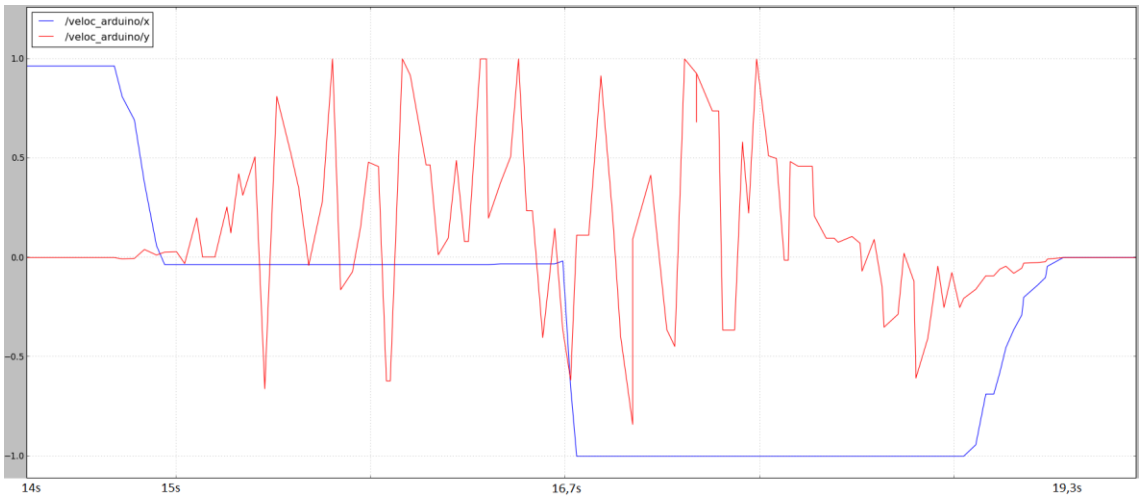
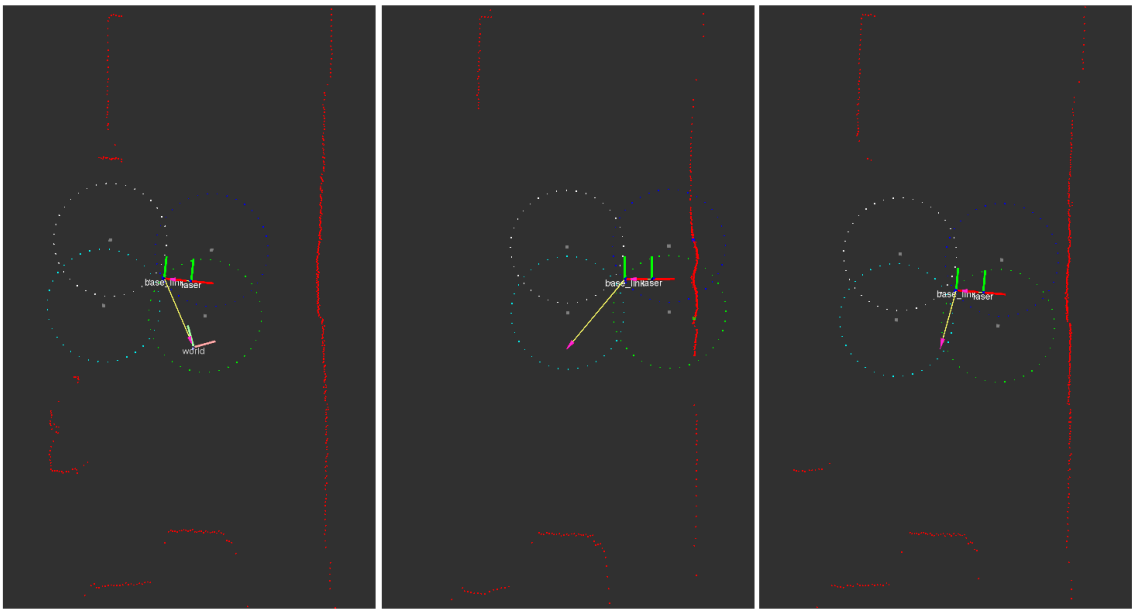


Figura 6.11: Gráfico dos sinais de velocidade gerados pelo nó *avoidance_teleop* diante de uma parede perpendicular.



a) b) c)

Figura 6.12: a) o sinal de velocidade para frente é dado no *joystick* b) a cadeira para devido ao cancelamento do sinal repulsivo da parede com o sinal do *joystick* c) o sinal do *joystick* cessa e a cadeira se desloca de ré devido ao sinal repulsivo até parar fora do campo repulsivo.

Na situação abaixo (Figura 6.13), a cadeira recebe comando de velocidade para frente e para a esquerda de 20s a 22s até que a parede fique oblíqua à cadeira. Em 24s apenas o comando para frente é dado, mas o sinal repulsivo da parede produz um torque no sentido horário até que a cadeira se alinhe com a parede em 28s.

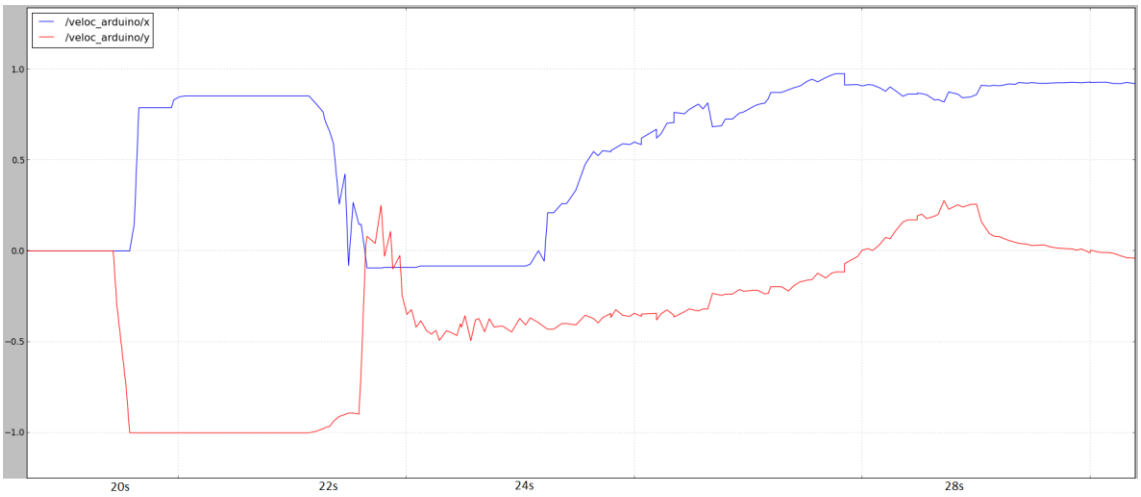
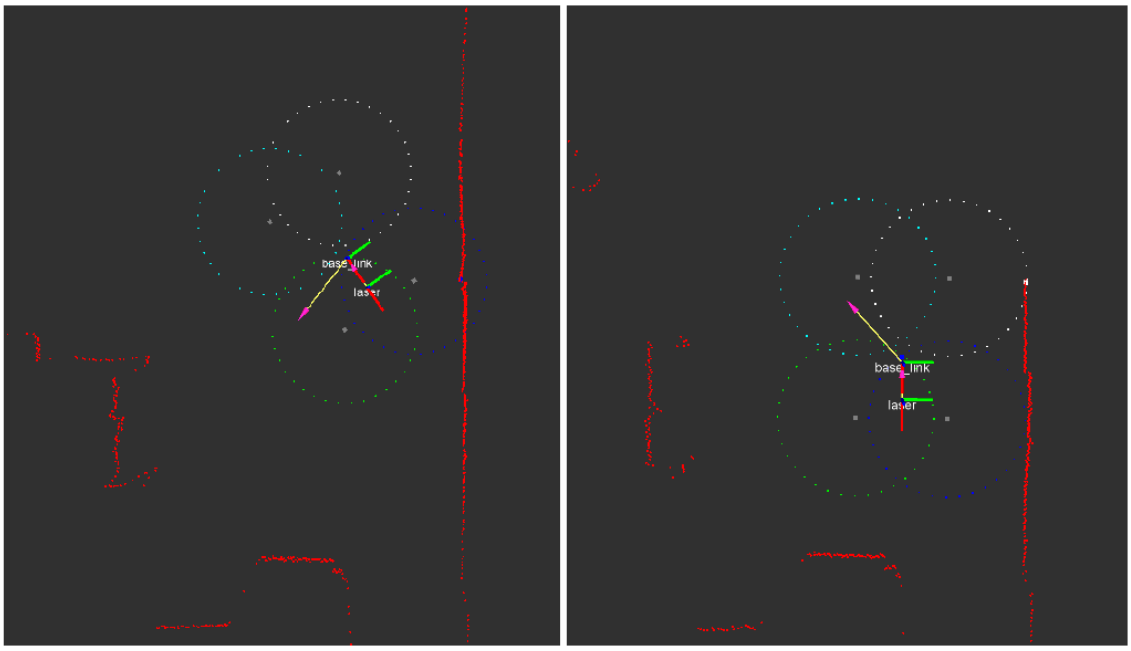


Figura 6.13: Gráfico dos sinais de velocidade gerados pelo nó *avoidance*_teleop diante de uma parede oblíqua.



a) b)

Figura 6.14: a) o sinal de velocidade para frente é dado no *joystick* b) o sinal repulsivo da parede atua sobre apenas um dos vértices da cadeira, gerando um torque que alinha a cadeira à parede.

6.7.2 Planejamento de trajetória:

Arquivo *Launch*: *run_apf.launch*

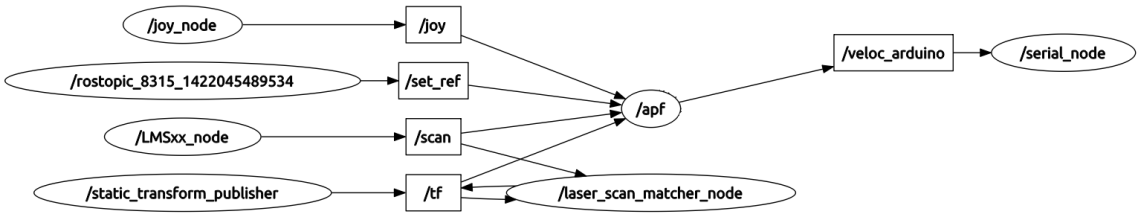


Figura 6.15: Diagrama de nós para o planejamento de trajetória.

A Figura 6.16 mostra o erro entre a configuração de referência e a configuração atual da cadeira (z é o erro de orientação) ao longo da trajetória com obstáculos. A configuração inicial da cadeira é $q_0 = [-0{,}605 \quad 0{,}424 \quad -0{,}256]^T$.

A referência $q_{ref} = \left[0{,}4 \quad 3{,}6 \quad \frac{\pi}{2}\right]^T$ é dada no instante 14,1s e o deslocamento começa. O erro de estado estacionário a partir de 42,7s é provocado pela zona morta da cadeira. O programa ainda gera um pequeno sinal de velocidade (Figura 6.17) para que a referência seja alcançada, mas o atrito das rodas impede que a cadeira se mova.

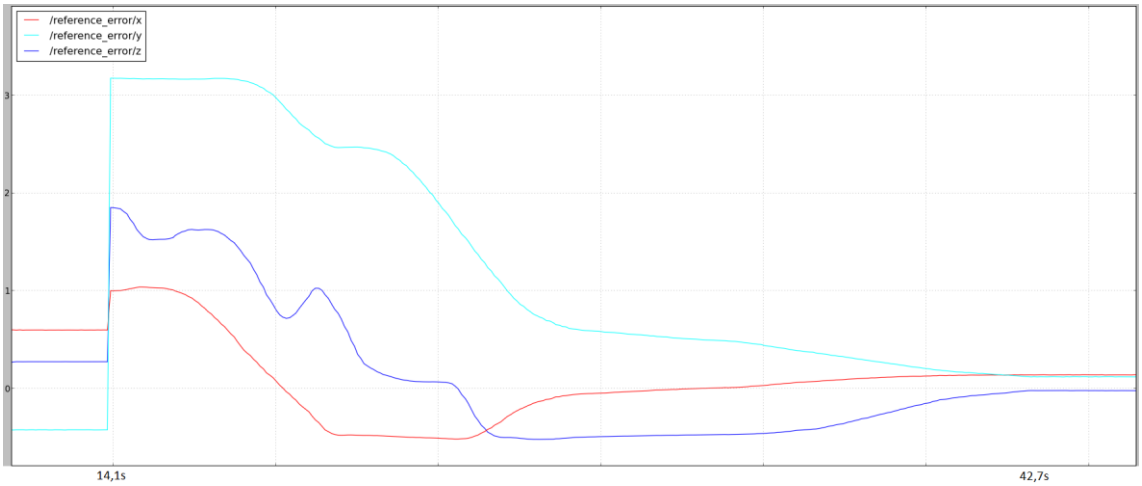


Figura 6.16: Gráfico do erro de configuração do robô até a referência.

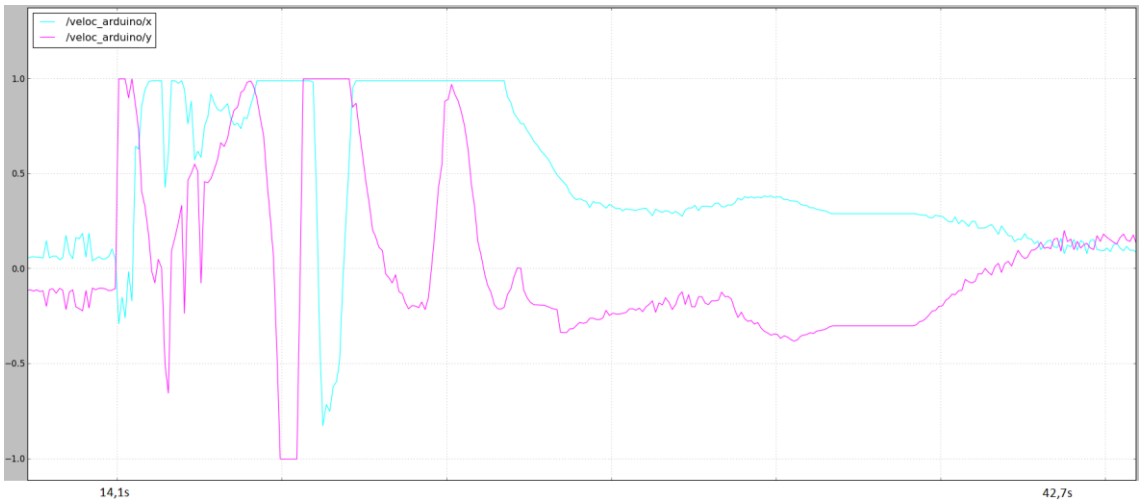
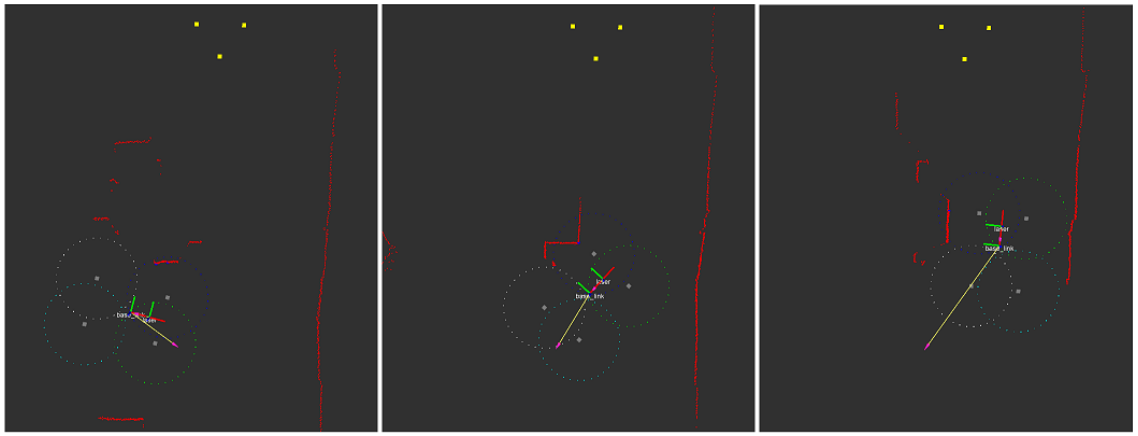
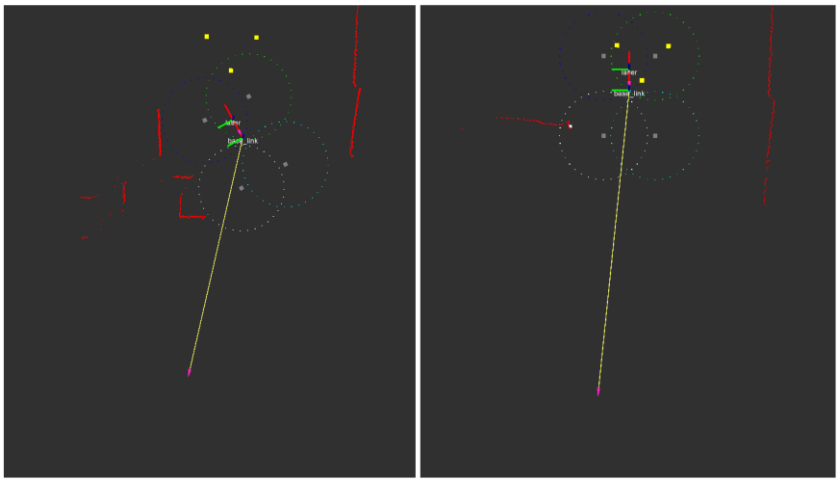


Figura 6.17: Gráfico dos sinais de velocidade gerados pelo nó *apf* até atingir a referência.



a) b) c)



d) e)

Figura 6.18: a) configuração inicial da cadeira b), c), d) a cadeira se aproxima da referência e) a cadeira para com um pequeno erro estacionário.

6.7.3 Deslocamento pelos corredores:

Arquivo *Launch*: *run_line_rec.launch*

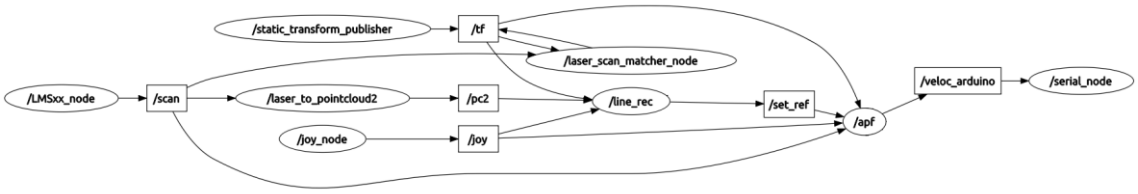


Figura 6.19: Diagrama de nós para o deslocamento pelos corredores.

As duas situações abaixo foram testadas em uma curva em T.

Na situação abaixo (Figura 6.20), o comando de velocidade para frente é dado em 9,7s, gerando uma referência constantemente atualizada para estar a 10m da cadeira (z é a orientação). Em 36s, o comando de parar é dado. As oscilações do sinal de velocidade angular, na Figura 6.21, tem amplitude pequena e não afetam o movimento reto da cadeira.

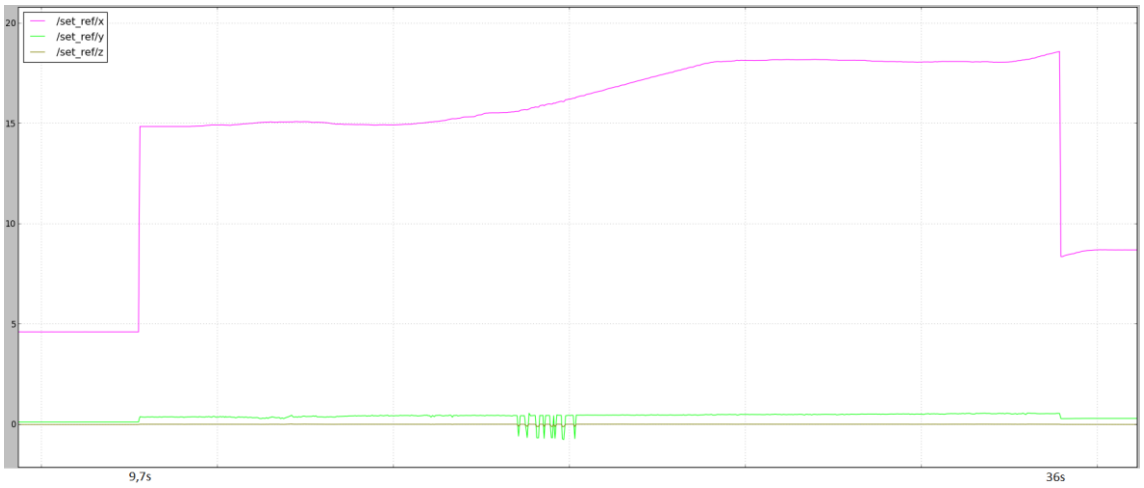


Figura 6.20: Gráfico da referência gerada pelo nó *line_rec* durante o deslocamento.

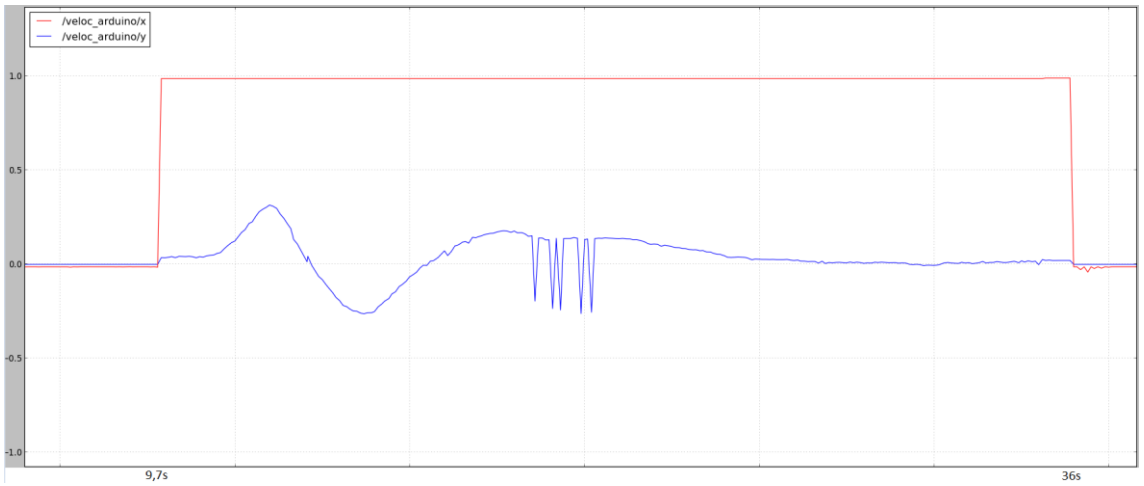
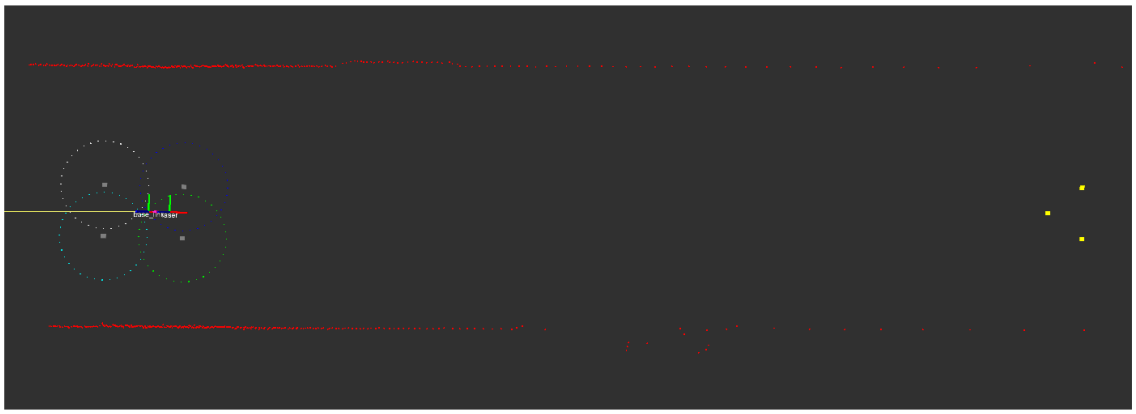
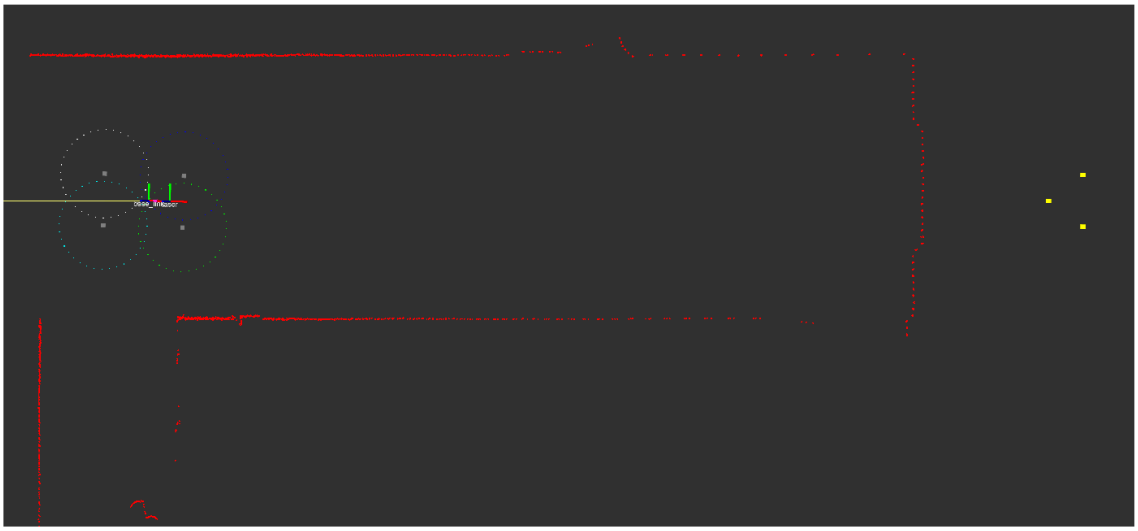


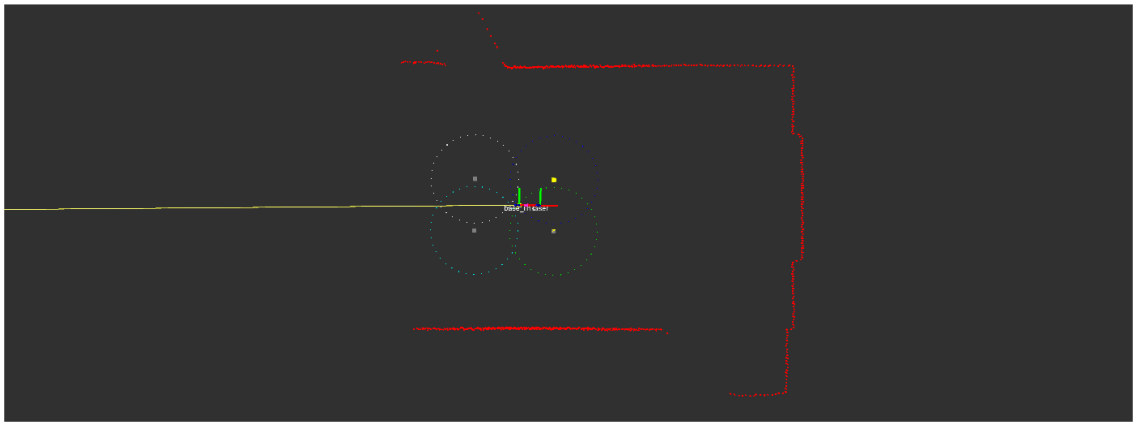
Figura 6.21: Gráfico dos sinais de velocidade gerados pelo nó *apf* durante o deslocamento.



a)



b)



c)

Figura 6.22: a) o comando de seguir em frente é dado e a referência é colocada à frente da cadeira b) a referência é constantemente atualizada para estar sempre à frente da cadeira c) o comando de parar é dado e a referência é colocada na configuração atual da cadeira.

Na situação abaixo (Figura 6.23) é dado o comando para a direita em 115s, mas a cadeira está em um corredor e a referência é colocada 10m à frente. Em 126,4s a cadeira encontra uma entrada à direita e a referência é colocada nessa direção, fazendo a cadeira iniciar a curva. O corredor no qual a cadeira entra é estreito e tem muitos objetos encostados na parede, tornando seu formato irregular, o que faz a cadeira andar de forma oscilatória [16]. Em 140,4s é dado o comando de parada.

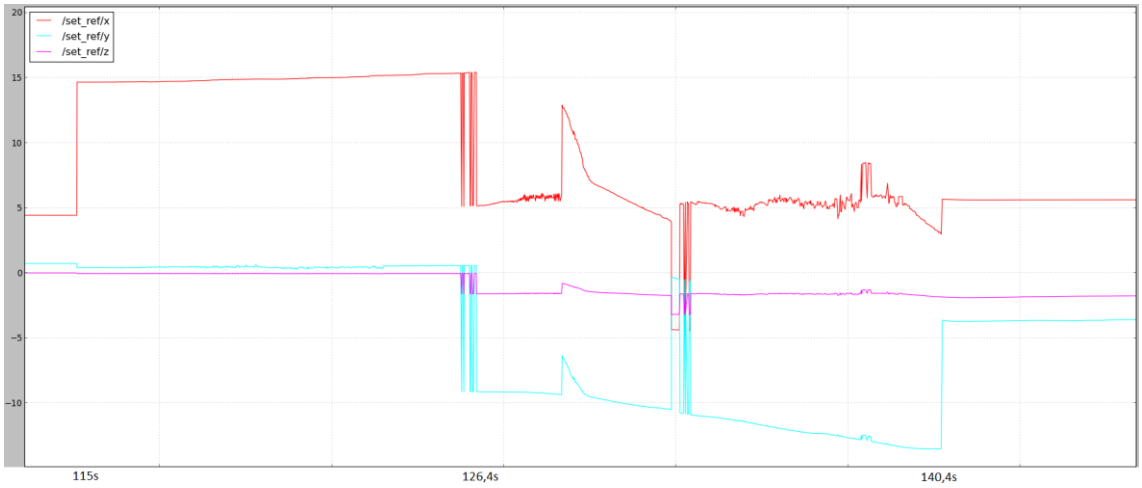


Figura 6.23: Gráfico da referência gerada pelo nó *line_rec* durante o deslocamento.

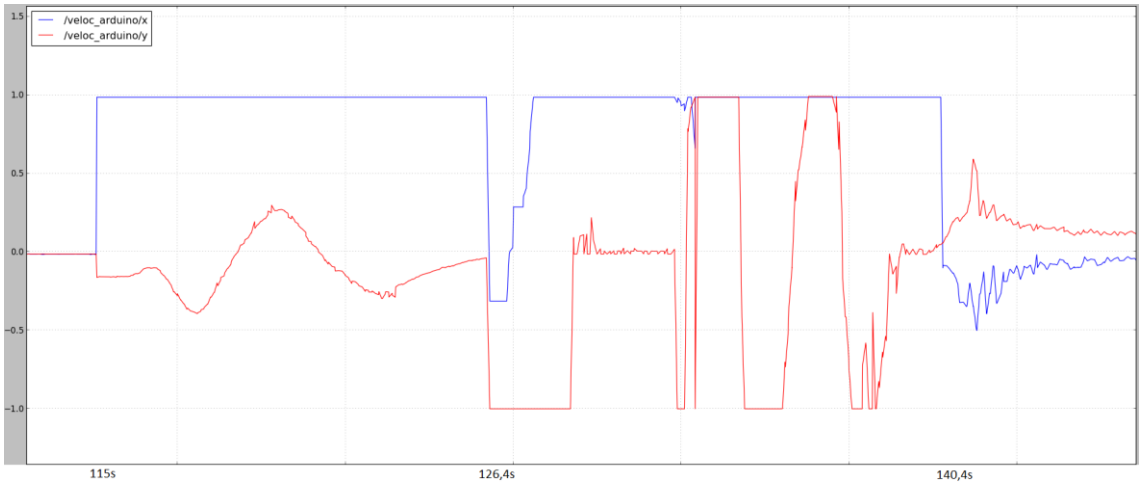
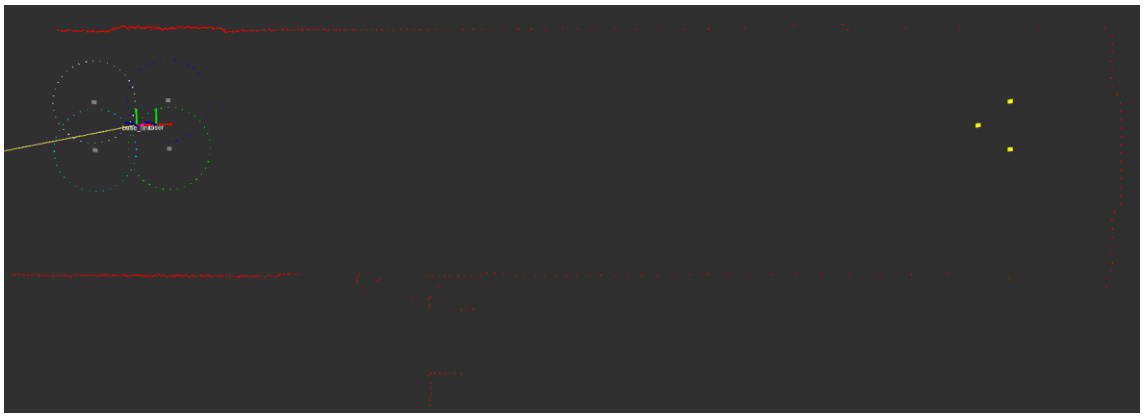
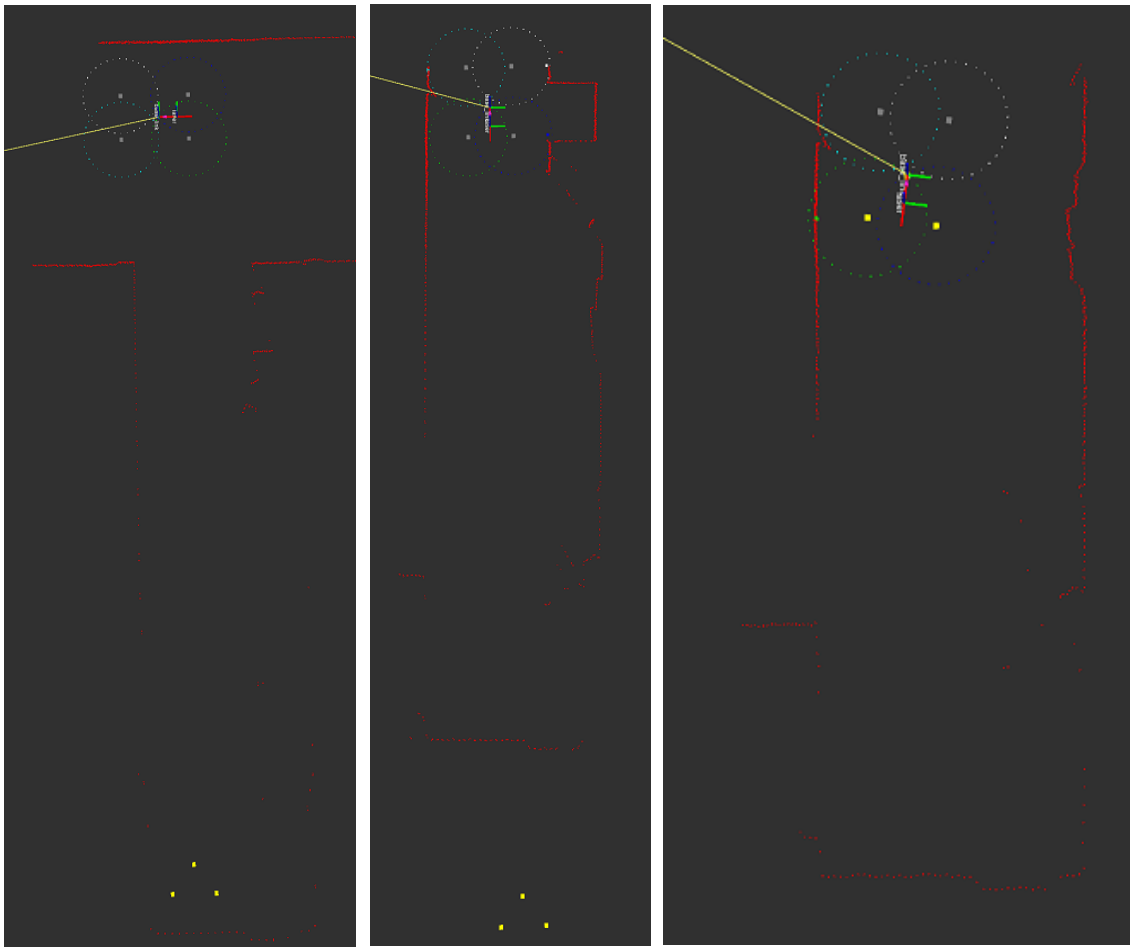


Figura 6.24: Gráfico dos sinais de velocidade gerados pelo nó *apf* durante o deslocamento.



a)



b) c) d)

Figura 6.25: a) o comando de virar no próximo corredor à direita é dado, mas, como a cadeira está em corredor, a referência é colocada a frente b) a cadeira encontra um corredor à direita e a referência é colocada nesse corredor c) a cadeira faz a curva e segue pelo corredor d) o comando de parar é dado.

6.8 Conclusões

No desenvolvimento deste trabalho, foi possível compreender e analisar a estratégia de planejamento de trajetória. Além dos conhecimentos adquiridos em navegação de robôs, principalmente em relação ao Método dos Campos Potenciais Artificiais, foram adquiridos conhecimentos em diversas etapas necessárias para a implementação desse algoritmo, como nas áreas de localização de robôs e extração de características a partir de dados dos sensores, que por si só possuem inúmeras outras aplicações, mesmo fora da robótica. As ferramentas utilizadas, principalmente o ROS e o Arduino, também possuem aplicações nas mais diversas áreas e produziram conhecimentos de eletrônica, sistemas operacionais, redes de computadores e programação.

Quanto aos resultados obtidos nos testes, verificamos que características indesejáveis dos equipamentos podem prejudicar o pleno funcionamento dos programas utilizados, principalmente a zona morta de sinal de controle da cadeira devido ao atrito das rodas da mesma. Esse erro, porém, é relativamente pequeno, sempre da ordem de alguns centímetros, pois é provocado por uma característica física, independente do APF. Tal erro somente é significativo para aplicações em que alta precisão seja necessária. Para esses casos, o APF pode não ser o algoritmo de navegação mais indicado, pois não gera uma trajetória baseado no modelo do robô, e portanto, essa zona morta é ignorada. A restrição não-holonômica de não-deslizamento lateral também pode afetar o desempenho do robô, já que reduz o seu conjunto de trajetórias factíveis, embora isso não tenha sido verificado nos resultados. Essa redução no conjunto de trajetórias teria efeito mais significativo caso buscássemos uma otimização de tempo. De modo geral, por não utilizar o modelo do robô, o APF apresenta desempenho superior em robôs holonômicos, para os quais não é há restrições de velocidade.

Outra limitação de equipamento se deve ao fato do *laser scanner* não captar pontos que estejam atrás da cadeira. Esse problema apenas exigiu o cuidado de não pôr a referência atrás da cadeira, de forma que ela não se deslocasse de ré. Já os ruídos nos sinais repulsivos devido às incertezas dos pontos captados pelo *laser scanner* não geraram prejuízo significativo ao desempenho da cadeira por causa da filtragem do circuito RC.

Os resultados foram considerados satisfatórios, cumprindo os objetivos propostos inicialmente por esse trabalho. O reconhecimento de corredores para navegação através de comandos simples, em particular, é uma aplicação prática essencial para portadores de deficiência física.

6.9 Trabalhos Futuros

Testar o Método de Campos Potenciais para outros tipos de robôs, como os que se deslocam no espaço, como quadricópteros, e também robôs articulados. Também seria interessante testar o APF em um robô terrestre holonômico, para verificar a diferença de desempenho em relação a essa cadeira de rodas não-holonômica. Devido à dificuldade de utilizar o APF para robôs com zona morta, outros algoritmos também podem vir a ser testados.

Bibliografia

[1] CHOSET, H. LYNCH, K. HUTCHINSON, S. KANTOR, G. BURGARD, W. KAVRAKI, L., THRUN, S. *Principles of Robot Motion: Theory, Algorithms and Implementation,* 1ª ed. The MIT Press, 2005.

[2] KHATIB, O., LE MAITRE, J. "Dynamic control of manipulators operating in a complex environment". *Proc. 3rd CISM-IFToMM Symp. Theory Practice Robots Manipulators*(1978), 267-282.

[3] SICILIANO, B., SCIAVICCO, L., VILLANI, L., AND ORIOLO, G. *Robotics: Modelling, Planning and Control*, 1ª ed. Springer, 2009.

[4] SIEGWART, R., NOURBAKHSH. *Introduction to Autonomous Mobile Robots,* 1ª ed. The MIT Press, 2004

[5] KHATIB, O. "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots". *The International Journal of robotics Research*(1986), 90-98.

[6] LATOMBE, J-C. *Robot Motion Planning,* 1ª ed. Kluwer Academic Publishers, 1991.

[7] CENSI, A. "An ICP variant using a point-to-line metric". *Proceedings of the IEEE International conference on Robotics and Automation(ICRA)*(2008), 19-25.

[8] BESL, P., MCKAY, N. "A method for registration of 3-D shapes". *IEEE Transactions on Pattern Analysis and Machine Intelligence*(1992), 239-256.

[9] HORN, B. "Closed-form solution of absolute orientation using unit quaternions". *Journal of the Optical Society of America*(1987), 629-642.

[10] RUSINKIEWICZ, S. LEVOY, M. "Efficient variants of the ICP algorithm". *Third International Conference on 3D Digital Imaging and Modeling (3DIM)*(2001).

[11] PRESS, W., TEUKOLSKY, S., VETTERLING, W., FLANNERY, B. *Numerical Recipes in C: The Art of Scientific Computing,* 2ª ed, Cambridge University Press, 1992.

[12] LOW K-L. "Linear Least-Squares Optimization for Point-to-Plane ICP Surface Registration". *Technical Report Department of Computer Science, University of North Carolina,* (2004).

[13] CHEN, Y., MEDIONI, G. "Object modeling by registration of multiple range images". *Proceedings of the IEEE International conference on Robotics and Automation (ICRA)*(1991), 2724-2729.

[14] POTTMAN, H., HUANG, Q-X. YANG, Y-L., HU, S-M. "Geometry and convergence analysis of algorithms for registration of 3d shapes". *International Journal of Computer Vision*(2006), 277-296.

[15] FISCHLER, M., BOLLES, R. "Random Sample Consensus: A Paradigm for Model Fitting with application to image analysis and automated cartography". *Communications of the ACM*(1981), 381-395.

[16] KOREN, Y., BORENSTEIN, J. "Potential Field Methods and Their Inherent Limitations for Mobile Robot Navigation". *Proceedings of the IEEE Conference on Robotics and Automation(ICRA)*(1991), 1398-1404.

Apêndice A:

Especificações Técnicas dos Equipamentos

Cadeira de Rodas Jazzy 600:

SPECIFICATIONS

Weight capacity	300 lbs.
Maximum speed	Up to 5.3 mph
Ground clearance	2.8"
Turning radius	22.5"
Overall length	35" without front riggings
Overall width	23"
Drive wheels	14"
Front casters	5" solid OMNI-casters
Rear casters	6" solid OMNI-casters
Suspension	Active-Trac® ATX Suspension
Drivetrain	Two-motor, Mid-Wheel 6®
Braking system	Intelligent braking (electronic, regenerative disc brakes)
Standard electronics	60 amp, PG VR2
Specialty controls	No
Battery charger	5 amp, off-board
Per-charge range	Up to 15.9 miles
Battery requirements	(2) 12 volt, deep cycle
Battery size	NF-22
Battery weight **Base weight** **Standard seat weight**	38 lbs. each 127 lbs. 37 lbs. (medium-back)
Max. seat dimensions	22" x 20"
Warranty	**K0822/K0823 Models:** Lifetime limited warranty on frame; 13-month limited warranty on electronics; 13-month limited warranty on drive motors **K0848/K0849/K0856/K0857 Models:** Lifetime limited warranty on frame; 2-year limited warranty on electronics; 18-month limited warranty on drive motors

MODELS

Jazzy 600 2S-SS	(Code: K0822)
Jazzy 600 2S-C	(Code: K0823)
Jazzy 600 3S-SS	(Code: K0848)
Jazzy 600 3S-C	(Code: K0849)
Jazzy 600 3SP-SS	(Code: K0856)
Jazzy 600 3SP-C	(Code: K0857)

FEATURES

- OMNI-Casters (nylon, spherical-shaped casters) on front and rear to prevent wheel hang-ups
- Side-mounted, easily accessible freewheel levers
- Active-Trac® ATX Suspension for enhanced performance over more varied terrain
- Easy rear access to batteries

OPTIONS

- Solid seat pan
- Cup holder
- Swing-away joystick
- Swing-away footrests
- TRU-Balance® Power Tilt (Group 3 only)
- Elevating leg rests
- Oxygen tank holder
- Cane/crutch holder
- Walker holder
- Synergy® Seat

RECOMMENDED LIFTS

Pride™ Lifts & Ramps

- Backpacker® AVP
- Backpacker® MV
- Backpacker® Plus
- Outlander
- PrideBoom 250

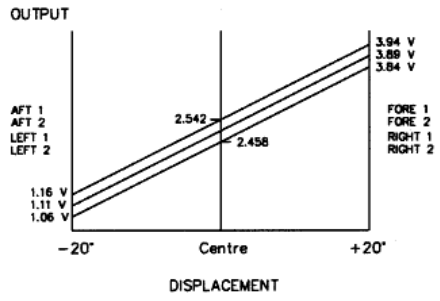
Driver de Potência da cadeira:

SPECIFICATION

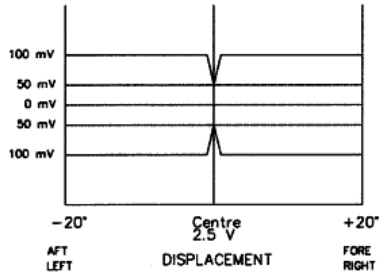
(@ 5 Vdc SUPPLY +20 °C AMBIENT TEMP.)
FOR FURTHER SPECIFICATION, CONSULT PG DRIVES TECHNOLOGY Ltd.

ELECTRICAL – supply	
OPERATING SUPPLY VOLTAGE (Vs)	5.0 Vdc ± 0.5 Vdc
ABS MAX VOLTAGE	24 V
REVERSE POLARITY PROTECTION	−18 Vdc
CURRENT CONSUMPTION	19 mA nominal
ELECTRICAL – outputs	
OUTPUT TYPE	Duplex – two independent per channel
OUTPUT IMPEDANCE	98 to 113 Ω
OUTPUT VOLTAGE SPAN	± 1.39 V wrt Vs/2 ± 50 mV
CURRENT (Sink/Source)	± 1.2 mA
RESOLUTION	Infinite
CENTRE ACCURACY	Vs/2 ± 42 mV
INDEPENDANT LINEARITY	1 % typ. (wrt 2.78 V)
CROSSTALK	< 4 % (wrt F.S.D.)
NOISE	1.5 mV typ.
ELECTRICAL – Vs/2	
CENTRE TAP VOLTAGE	50 % of Vs ± 1.2 %
CENTRE TAP IMPEDANCE	1100 Ω ± 1 %
MECHANICAL	
LEVER OPERATING FORCE	
BREAKOUT	1.0 N nominal (@ 2° deflection)
OPERATING	1.3 N nominal (@ 20° deflection)
MAX. LOAD ALLOWABLE	300 N
MECHANICAL ANGLE	
ROUND GATE	± 20° nominal
MAX. SHAFT TORQUE ALLOWABLE	< 2 Nm
EXPECTED LIFE	15 million operations
WEIGHT	0.076 kg nominal
ENVIRONMENTAL	
OPERATING TEMPERATURE	−25° C TO +70° C
STORAGE TEMPERATURE	−40° C TO +70° C
SEALING (ABOVE FLANGE)	IP65 to IEC 60529:1991
EMC – IMMUNITY	IEC 61000-4-3:2002 50 V/m
EMC – EMISSIONS	BS EN 55022:1998, Class B,
ESD IMMUNITY	IEC 61000-4-2:2001 15 kV air, 8 kV contact

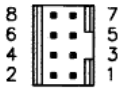
OUTPUT VOLTAGE Vs DISPLACEMENT



DIFFERENTIAL OUTPUT (SAME CHANNEL)



CONNECTION DETAILS

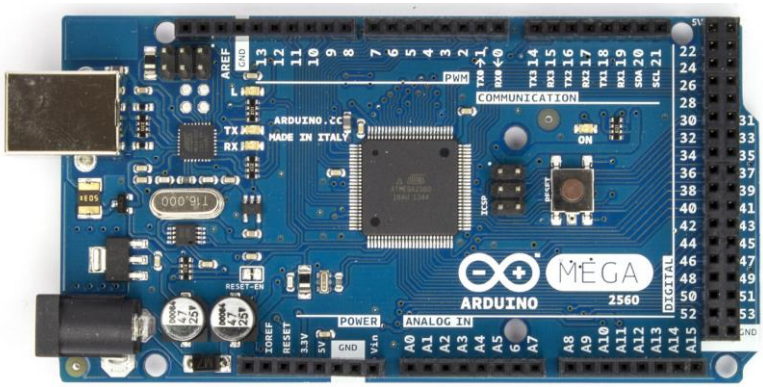


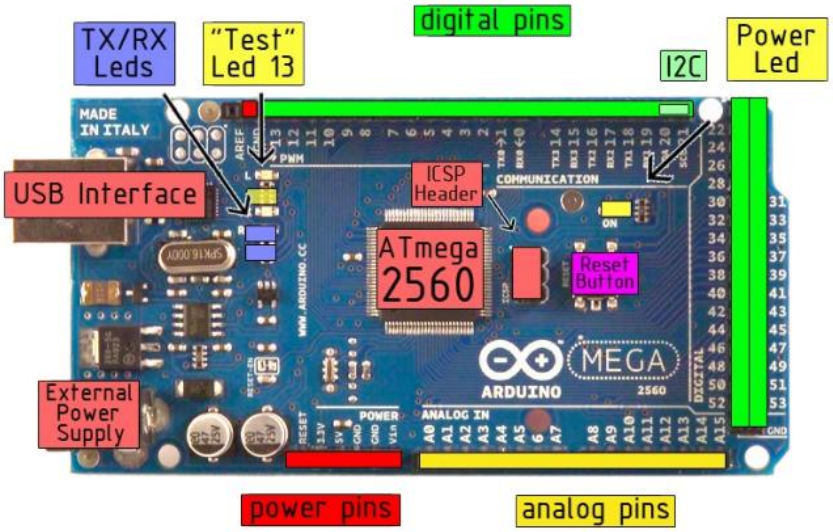
Connector Details

Connector Type	FCI Minitek	90309-008
Mating IDC Connector	FCI Minitek	89361-708

Pin Number	Description
1	Voltage Supply Vs (5 Vdc)
2	Left/Right output 1
3	Voltage Supply 0 V
4	Fore/Aft output 1
5	Fore/Aft output 2
6	Centre tap voltage Vs/2
7	Left/Right output 2
8	Not connected

Arduino Mega 2560





Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	54 (of which 14 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz

Controle Xbox 360

5.3 Microsoft Xbox 360 Wired Controller for Linux

Table of index number of `/joy.buttons`:

Index	Button name on the actual controller
0	A
1	B
2	X
3	Y
4	LB
5	RB
6	back
7	start
8	power
9	Button stick left
10	Button stick right

Table of index number of `/joy.axis`:

Index	Axis name on the actual controller
0	Left/Right Axis stick left
1	Up/Down Axis stick left
2	LT
3	Left/Right Axis stick right
4	Up/Down Axis stick right
5	RT
6	cross key left/right
7	cross key up/down

Laser Scanner SICK LMS111-10100

Features	
Field of application:	Outdoor
Version:	Short Range
Light source:	Infrared (905 nm)
Laser class:	1 (IEC 60825-1 (2007-3))
Field of view:	270 °
Scanning frequency:	25 Hz/50 Hz
Angular resolution:	0.25 ° 0.5 °
Heating:	yes
Operating range:	0.5 m ... 20 m
Max. range with 10 % reflectivity:	18 m
Amount of evaluated echoes:	2
Fog correction:	yes

Performance	
Response time:	≥ 20 ms
Detectable object shape:	Almost any
Systematic error:	± 30 mm
Statistical error:	± 12 mm
Integrated application:	Field evaluation
Number of field sets:	10 fields
Simultaneous processing cases:	10

Interfaces	
Serial (RS-232):	✓
Function (Serial (RS-232)):	Host, AUX
Data transmission rate (Serial (RS-232)):	9.6 kBaud ... 115.2 kBaud
Ethernet:	✓
Function (Ethernet):	Host
Data transmission rate (Ethernet):	10/100 Mbit
Protocol (Ethernet):	TCP/IP, OPC
CAN bus:	✓
Function (CAN bus):	Outputs extension
PROFIBUS:	-
PROFINET:	-
DeviceNet:	-
Switching inputs:	2
Switching outputs:	3
Optical indicators:	1 7-segment display (plus 5 LEDs showing device status, contamination warning and initial condition)

Mechanics/electronics	
Electrical connection:	M12 circular plug-in connector
Operating voltage:	10.8 V DC ... 30 V DC
Power consumption:	60 W
Housing color:	gray (RAL 7032)
Enclosure rating:	IP 67 (EN 60529, Section 14.2.7)
Protection class:	III (EN 50178 (1997;10))
Weight:	1.1 kg, without connecting cables
Dimensions:	105 mm x 102 mm x 162 mm

Ambient data	
Object remission:	2 % ... > 1,000 % (reflectors)
Electromagnetic compatibility (EMC):	EN 61000-6-2:2005/EN 61000-6-4 (2007-01)
Vibration resistance:	EN 60068-2-6 (1995-04)
Shock resistance:	EN 60068-2-27 (1993-03)
Ambient operating temperature:	-30 °C ... 50 °C
Storage temperature:	-30 °C ... 70 °C
Ambient light safety:	40,000 lx

Operating range diagram