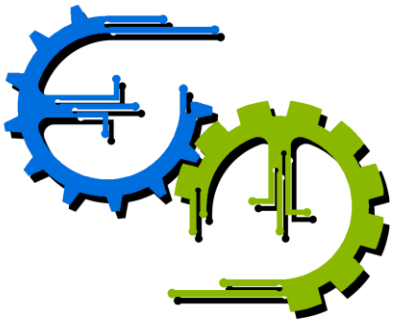
TRABALHO DE GRADUAÇÃO

Integração entre Planejamento e Execução de Rotas para um Manipulador Robótico Móvel

Waldez Azevedo Gomes Júnior

Brasília, julho de 2014



ENGENHARIA
MECATRÔNICA
UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia

TRABALHO DE GRADUAÇÃO

Integração entre Planejamento e Execução de Rotas para um Manipulador Robótico Móvel

Waldez Azevedo Gomes Júnior

Relatório submetido como requisito parcial para obtenção do grau de Engenheiro de Controle e Automação

Banca Examinadora

Prof. Geovany Araújo Borges, ENE/UnB _____
Orientador

Prof. Antônio Padilha Lanari Bó, ENE/UnB _____
Examinador interno

Prof. João Yoshiyuki Ishihara, ENE/UnB _____
Examinador interno

FICHA CATALOGRÁFICA

WALDEZ, AZEVEDO GOMES JÚNIOR Integração ente Planejamento e Execução de Rotas para um Manipulador Robótico Móvel, [Distrito Federal] 2014. xvi, 74p., 297 mm (FT/UnB, Engenheiro, Controle e Automação, 2014). Trabalho de Graduação – Universidade de Brasília. Faculdade de Tecnologia.
1.Direção Diferencial — 2. Manipulação Móvel 3. Planejamento de Rotas — 4. Execução de Rotas I. Mecatrônica/FT/UnB — II. Título (Série)

REFERÊNCIA BIBLIOGRÁFICA

GOMES JUNIOR, W. A. (2014). Integração entre Planejamento e Execução de Rotas para um Manipulador Robótico Móvel. Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT.TG-$n°$05/2014, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 74p.

CESSÃO DE DIREITOS

AUTOR: Waldez Azevedo Gomes Júnior

TÍTULO DO TRABALHO DE GRADUAÇÃO: Integração entre Planejamento e Execução de Rotas para um Manipulador Robótico Móvel

GRAU: Engenheiro ANO: 2014

Waldez Azevedo Gomes Júnior
SQN 407 Bloco F Apto 203 - Asa Norte
70855-060 - Brasília - DF - Brasil

Dedicatória

Dedico este trabalho de graduação especialmente a minha mãe Valdires e meu pai Waldez que sempre me apoiaram em tudo na minha vida e que certamente são os pilares de construção de tudo que há de bom na minha vida hoje.
Dedico também às minhas três irmãs, Crisane, Carolina, e Cristiane que frequentemente são as únicas pessoas capazes de me compreender.
Te amo pessoal!

Waldez Azevedo Gomes Júnior

Agradecimentos

Agradeço primeiramente a todos meus colegas de curso da engenharia mecatrônica, principalmente a uma turminha do barulho que apronta altas confusões, os meus grandes amigos Fabinho, Arno, Ricardo, André, Ivo, Glauco, João, Luiz e demais integrantes da melhor turma que a mecatrônica há de ter, a turma 23. Por causa do intercâmbio, passei quase dois semestres do curso sem vocês e não foi nem perto de ser a mesma coisa.
Agradeço ao pessoal da AMORA, que sempre estavam ali pra me ajudar quando eu mais precisava durante o semestre; ao George pelos projetos em que participei, assim como pelos vários conselhos dados durante todo o curso; ao Miguel, que me deu valiosas lições de SolidWorks *num momento crítico do TG; a Renata que aos 45 do segundo tempo fez questão de me ajudar com a revisão do TG.*
Agradeço a todos meus professores na UnB. Acredito profundamente que todos eles têm uma parcela de responsabilidade grande no desenvolvimento do engenheiro Waldez.
Agradeço a meus professores do tempo de ensino fundamental e médio também. Especialmente às aulas da professora Dalvani que tanto me fizeram gostar de história, de contar histórias. E às aulas de matemática do professor Romildo que me ajudaram a ver um potencial que eu nunca havia percebido de verdade em mim. Eu não estaria fazendo um trabalho de graduação não fossem todos eles.

Waldez Azevedo Gomes Júnior

RESUMO

Existem diversas plataformas robóticas móveis, que possuem manipuladores, usadas tanto em ambiente industrial quanto acadêmico. Para um uso eficiente desses robôs é necessário que se faça um planejamento das rotas a serem executadas pelos robôs. Nesse trabalho, primeiramente foram estudadas técnicas de planejamento de rotas para espaços de configurações genéricos. Dentro dos vários tipos de algoritmos para planejamento, foram escolhidos e estudados mais a fundo algoritmos de planejamento que usam uma amostragem probabilística do espaço de configurações. Foi feita uma simulação de uma plataforma robótica com uma base móvel com direção diferencial e um braço robótico com sete graus de liberdade e uma garra. Também foi realizada a integração entre planejadores de rotas e módulos de execução das rotas no robô simulado (desenvolvidos especialmente para essa aplicação). Foram realizados testes para definir quais os melhores algoritmos de planejamento de rotas tanto para a base móvel do robô quanto para o braço robótico. Os experimentos dos módulos de planejamento e execução mostraram que esses módulos ainda podem ser melhorados para se obter uma aplicação de melhor qualidade. Apesar disso, a integração entre planejamento e execução de rotas foi realizada com sucesso.

Palavras Chave: Direção diferencial, manipulação móvel, planejamento de rotas, execução de rotas.

ABSTRACT

There are various kinds of mobile robot platforms with built-in manipulators that are used in industrial and academic environments. In order to use such mobile manipulators, one needs to plan beforehand the paths that are going to be executed by the robot. In this work, there is a summarization of path planning techniques for generic configuration spaces. Among various planning algorithms, we chose sampling based planners for use with our application. It was developed a simulation for a robotic platform which has a mobile base with a differential drive, and a robotic arm with seven degrees of freedom, plus a gripper. Furthermore, the path planning modules were integrated with path execution modules which were specially developed for this application. There were made tests for the path planning algorithms for the mobile base and for the manipulator in order to decide which algorithms performed better in our robot simulation. Despite of the experiments that showed that the planning and execution modules still have a lot of room to improve, the integration between planning and execution for our mobile manipulator was successfully done.

Keywords: Differential drive, mobile manipulation, path planning, path execution.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

Símbolos Latinos

$\mathbb{R}$	Conjunto dos números reais
$\mathcal{C}$	Espaço de configurações
$\mathcal{W}$	Ambiente onde o robô se encontra
H	Semiplano
n	Nó de uma *OcTree*
P	Probabilidade
q	Configuração de um robô
$\mathcal{O}$	Espaço topológico de obstáculos para o robô
$\mathbb{Z}$	Conjunto dos números inteiros
z	Medida de profundidade
l	Número de observações para atualização de estado do voxel / distância entre rodas
$\mathcal{A}$	Espaço topológico do robô
T	Matriz de transformação homogênea
R	Matriz de rotação
v	Vetor de rotação para um quatérnio / velocidade linear de um robô com direção diferencial
$\mathbb{H}$	Grupo dos quatérnios
$\mathbb{S}^1$	Espaço topológico para um círculo
$SO(n)$	Grupo de matrizes de rotação n-dimensional
$SE(n)$	Grupo especial euclideano
r	Raio das rodas
$\mathcal{G}$	Grafo
V	Vértice em um grafo
E	Aresta em um grafo
U	Espaço de ações de controle
DC	*Direct Current*

Símbolos Gregos

λ	Intervalo fechado
ξ	Configuração de um robô em $SE(2)$
φ	Posição no eixo de uma roda
$\mathcal{T}$	Árvore do tipo RRT
θ	Orientação de um robô em $SE(2)$
ρ	Distância em um espaço métrico
τ	Rota

Sobrescritos

$\dot{}$	Derivada de primeira ordem em relação ao tempo
$^{-1}$	Inversa da matriz
^{T}	Transposta da matriz
$\hat{}$	Estimativa
*	Quatérnio conjugado

Siglas

LARA	Laboratório de Robótica e Automação
MatLab	*Matrix Laboratory*
ODE	*open dynamics engine*
PC	*Personal Computer*
SLAM	*Simultaneous Localization and Mapping*
UnB	Universidade de Brasília
OMPL	*Open Motion Planning Library*
ROS	*Robot Operating System*
MSV	Método de Seleção de Vértice
MPL	Método de Planejamento Local
RRT	*Rapid Random exploring Tree*
AMORA	*Autonomous MObile Robot Algorithms*
RGB	Esquema de cores *Red Green Blue*
P2P	Topologia de rede *Peer-to-peer*
TF	Pacote do ROS chamado *Transformation Frame*
OGRE	*Object-oriented Graphics Rendering Engine*
URDF	*Unified Robot Description Format*
SDF	*Simulation Description File*
SRDF	Semantic Robot Description Format
STL	Arquivo do tipo *STereoLithography*
ASM	Arquivo do tipo *ASseMble*
RViz	Robot Visualization
PID	Proporcional Integral Derivativo
ACM	*Auto-Collision Matrix*
PRM	*Probabilistic RoadMap*
EST	*Expansive Space Tree*
K-PIECE	*Kinodynamic Motion Planning by Interior–Exterior Cell Exploration*

Capítulo 1

Introdução

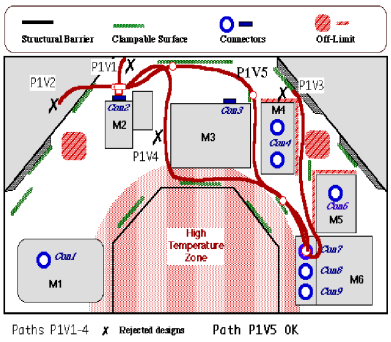
1.1 Contextualização

Construir autômatos capazes de realizar tarefas cotidianas, máquinas que interajam com o ambiente ao seu redor e principalmente com humanos faz parte do imaginário da sociedade moderna.

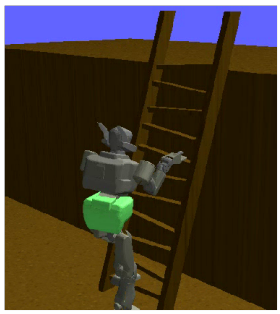
Um dos primeiros passos para que tais máquinas possam interagir com o ambiente é que elas possam se localizar, e mais ainda, identificar o que está ao seu redor a fim de que se possa ter conhecimento sobre aonde é possível se movimentar no espaço. Esses são problemas mais conhecidos como problemas de localização e mapeamento simultâneo, ou em inglês, *Simultaneous Localization And Mapping* (SLAM).

Não basta para a máquina saber onde está e aonde ela pode ir, ela precisa planejar *a priori* como irá se mover, o que pode ser uma tarefa difícil cujo grau de dificuldade pode ser ainda mais elevado devido a perturbações no ambiente de trabalho do robô, que em muitos casos são pessoas e objetos se movendo no espaço.

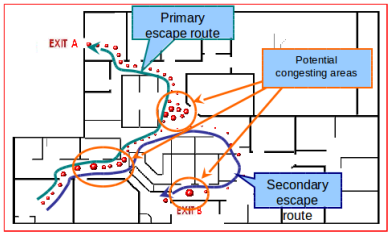
Ao realizar uma estratégia de movimento é calculado um planejamento de movimento que atinge um objetivo. Esse objetivo pode ser definido com base em vários tipos de requisitos, como



(a) Planejamento de canos em um ambiente industrial

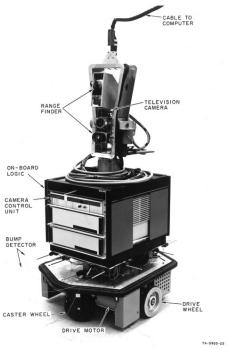


(b) Robôs Humanóides

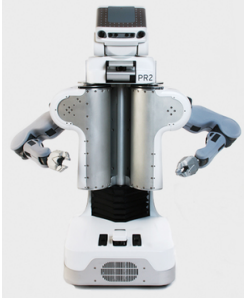


(c) Simulação de rotas de fuga em situações de emergência

Figura 1.1: Aplicações de planejamento de rotas



(a) *Shakey* - Robô desenvolvido na Universidade de Stanford entre os anos de 1966 e 1972



(b) PR2 - Robô desenvolvido pela *Willow Garage* para pesquisa acadêmica [8]

Figura 1.2: Exemplares de robôs móveis

requisitos geométricos, de tempo, de trajetória, etc.

Existem diversas aplicações (Fig. 1.1) para algoritmos que realizam planejamento de rotas em vários setores da indústria [2], tais como exploração de ambientes por robôs móveis, planejamento de cirurgias, animação de atores virtuais, projeto de distribuição de canos, ou de estruturas sensíveis a fatores externos, além de robôs humanóides. E muitas dessas aplicações utilizam manipuladores.

A manipulação de objetos é a maneira mais intuitiva para humanos de como interagir com o mundo e seus objetos. É natural a extensão desse conceito de manipulação para robôs. Além disso, muitos ambientes onde se faz necessária a automação de processos, como linhas de montagem, requerem o uso de manipuladores para processar objetos de variados tamanhos e formas.

Robôs móveis, desde o *shakey*, um dos primeiro robôs com base móvel (Fig. 1.2a), apresentam muitos desses desafios impostos ao estudo da robótica. Hoje em dia, existem iniciativas muito mais sofisticadas, que incluem manipuladores à base móvel, como o PR2 (Fig. 1.2b) da WillowGarage.

O LARA (Laboratório de Automação e Robótica) também possui um manipulador móvel, mas que no momento não usufrui de módulos de planejamento e execução de rotas feitos especialmente para ele.

1.2 Objetivo

O objetivo principal desse trabalho é o desenvolvimento de uma plataforma de simulação, que encapsula módulos de planejamento e execução de rotas para um manipulador robótico móvel, que poderá ser utilizada futuramente para avaliar algoritmos de planejamento de rotas baseados em amostragem do espaço de configurações do robô. Será considerado o robô em ambiente fechado onde será conhecida *a priori* a representação geométrica de objetos fixos.

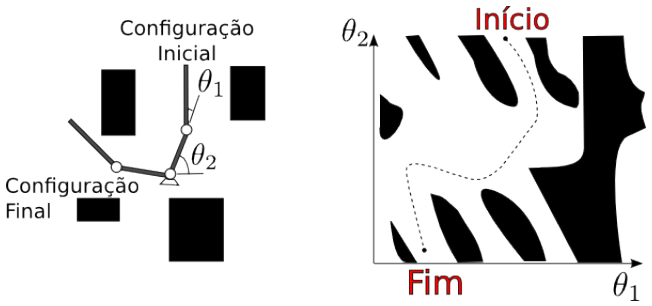


Figura 1.3: Braço robótico com duas dimensões, e uma trajetória no espaço de configurações [1]

1.3 Definição do Problema de Planejamento de Rotas

Para que se possa desenvolver aplicações para planejamento de rotas para um robô faz-se necessário que haja uma definição do problema de planejamento de rotas para robôs.

Um robô pode ser representado por uma configuração, que pode ser livremente definida como um vetor que contém informações sobre o atual estado do robô. O conjunto de todos os estados possíveis para um robô é então um espaço de estados especial chamado de espaço de configurações, $\mathcal{C}$.

Cada variável de estado geralmente é relacionada a um grau de liberdade do robô no ambiente em que se encontra. Esse ambiente é chamado de $\mathcal{W}$.

A representação do robô usando o espaço de configurações é bastante conveniente pois permite representar robôs com formas complexas como um simples ponto de várias dimensões.

O planejamento de rotas é a obtenção de pontos intermediários em $\mathcal{C}$, entre a configuração inicial e a configuração final desejada para o robô. Esses pontos intermediários definem uma sequência de ações que o robô deve executar.

Essa definição é muito parecida com um problema de resolução de uma equação diferencial com valores de contorno, no entanto, para a maioria dos casos não existe uma solução direta para o problema, devido principalmente a obstáculos no ambiente em que o robô se encontra.

A figura 1.3 mostra um exemplo de planejamento de rota para um braço robótico com dois graus de liberdade.

No contexto de manipulação de objetos, os objetos em questão podem ser anexados à forma do robô para que possa ser planejada uma trajetória levando em consideração o objeto. Isto é, quando o objeto é anexado ao robô, a forma do robô é estendida até a forma do objeto.

A anexação é feita logo após o momento em que o manipulador segura o objeto. Esse processo de segurar o objeto depende muito das características do objeto e do manipulador e consiste em um problema separado ao de planejamento de rotas, embora fique claro que para uma manipulação efetiva os dois problemas devem ser resolvidos preferencialmente de forma simultânea.

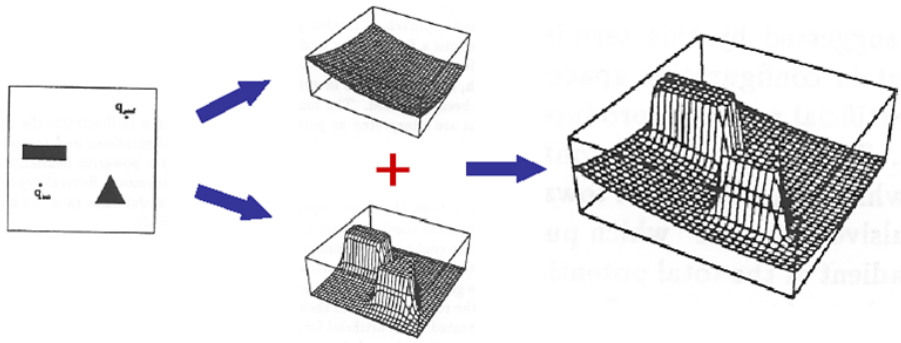


Figura 1.4: A figura mostra dois campos artificiais produzidos pelos obstáculos e pelo objetivo final, resultando num campo potencial artificial que pode ser usado para navegar o robô para seu destino.[2]

1.3.1 Tipos de planejadores

Existem várias classes de algoritmos de planejamento de rotas que podem ser aplicadas em robótica. Algumas dessas classes são introduzidas nas seções a seguir.

1.3.1.1 Campos potenciais artificiais

Esse método trata o robô como um ponto sob influência de um campo potencial artificial. Obstáculos repelem o robô, ao passo que o objetivo final , o local aonde o robô deve se dirigir, atrai o robô. Das interações entre obstáculos e objetivos guiados por uma função de navegação resulta um campo potencial artificial (Fig. 1.4) . O robô é atraído para o objetivo, assim como uma bola rolaria estando em cima de um morro [1].

É um método de fácil implementação e facilmente escalável para uma aplicação em tempo real, no entanto sofre bastante com mínimos locais dentro do campo (nessa abordagem o objetivo final é sempre o mínimo global do campo). É possível encontrar funções de navegação que garantam que não haja mínimos locais para espaços de configurações com poucas dimensões, no entanto, essa não é uma tarefa trivial.

1.3.1.2 Decomposição aproximada e exata de células

É possível particionar o ambiente $\mathcal{W}$ em células discretas que correspondam a espaços livres de colisão do robô com objetos (Fig. 1.5). Após essa partição pode se pensar no problema de planejamento como um problema de mover o robô de uma partição livre para outra até chegar ao destino final, dessa maneira pode-se modelar um grafo cujos vértices sejam as células e as arestas indicam adjacências entre as células. Nessa abordagem, o planejamento é resolvido fazendo uma busca nesse grafo.

A desvantagem desse método é que não é facilmente escalável para maiores dimensões de $\mathcal{C}$, tampouco para robôs com complexas não-linearidades de movimento. Entretanto é bastante útil em ambientes esparsos que contenham somente um robô e que todos os obstáculos estejam fixos.

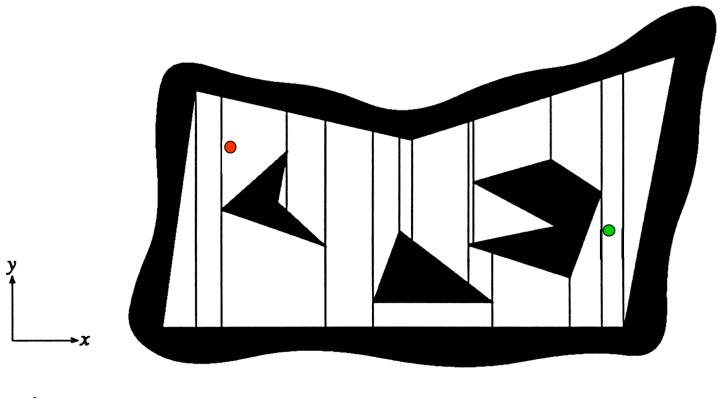


Figura 1.5: A figura mostra a decomposição de um $\mathcal{W}$ em células.[2]

1.3.1.3 Métodos baseados em teoria de controle

Baseado em uma modelagem correta das equações de movimento, é possível aplicar técnicas de teoria de controle com realimentação que usam o objetivo como entrada, e tentam levar o erro de posição à zero [1, 9, 3].

Essa abordagem tem uma resposta muito rápida, mas por vezes é muito difícil calcular uma solução para ambientes com dinâmica complexa, ou com muitos obstáculos. No entanto, é bastante utilizada como um planejador local de movimento entre dois pontos que não possuem obstáculos entre eles.

1.3.1.4 Planejamento Probabilístico

Abordagens não-determinísticas podem ser usadas para tirar o robô de mínimos locais em campos artificiais, ou para melhor modelar as células de espaço livre do robô. E também pode ser usada em um *framework* de planejamento de rotas baseado em amostragens de $\mathcal{C}$.

Essas técnicas amostram o espaço de configurações do robô aleatoriamente em uma distribuição uniforme para o caso mais simples, e conectam essas amostras através de caminhos livres de colisão que respeitam restrições impostas ao movimento do robô. Essas restrições podem estar relacionadas a obstáculos, a limitações físicas do robô, ou no contexto de manipulação a movimentos aceitáveis para uma peça que esteja sendo carregada pelo manipulador.

Devido ao fato de que manipuladores móveis possuem vários graus de liberdade, o uso de algoritmos dessa classe possuem melhor performance ao calcular rotas para manipuladores do que algoritmos determinísticos e completos cujo tempo de execução aumenta exponencialmente com o número de graus de liberdade de forma a ser impraticável o seu uso em manipuladores [10, 11, 12].

1.4 Contribuição do trabalho

Nesse trabalho foi realizada uma simulação computacional de uma plataforma robótica existente no LARA (Laboratório de Robótica e Automação) que possui um braço robótico com sete graus

de liberdade montado sobre uma base móvel. Além disso, foram realizados testes de algoritmos de planejamento de rotas no robô simulado.

Foram planejadas rotas para base móvel do robô e para o braço robótico separadamente, com algoritmos de planejamento baseados em amostragem do espaço de configurações do robô. Também foram implementadas estratégias de execução de rotas pelo robô.

1.5 Organização do manuscrito

No capítulo 2 é realizada uma fundamentação teórica de conceitos necessários para melhor compreensão do problema. Ele cobre desde fundamentações matemáticas até uma definição mais formal do problema de planejamento de rotas.

O capítulo 3 trata sobre o desenvolvimento do trabalho e para isso faz primeiro uma descrição da plataforma robótica existente no LARA, na qual o robô simulado é baseado.

É feito um breve resumo sobre as ferramentas computacionais mais importantes para o trabalho como o simulador Gazebo, a biblioteca de planejamento OMPL e o pacote de planejamento e execução de rotas MoveIt! [13, 14, 15] enquanto é explicada a importância de cada ferramenta e como é feita sua integração ao projeto.

A descrição do *framework* ROS é feita detalhada e cuidadosamente visto que o ROS é o ambiente de integração das ferramentas usadas para o projeto e quase sempre é necessário recorrer a conceitos relacionados ao ROS para falar sobre o trabalho.

O capítulo 4 mostra os principais resultados relativos ao trabalho desenvolvido. São mostrados resultados de testes de execução das rotas calculadas pelos planejadores, além de testes para vários algoritmos de planejamento de rotas.

No capítulo de anexos, são detalhados o conteúdo do CD entregue, uma descrição mais detalhada da plataforma robótica contendo desenhos técnicos e uma representação do robô no ROS, assim como os programas utilizados para a realização deste trabalho de graduação.

Capítulo 2

Fundamentação teórica

2.1 Introdução

Para que possa ser feito o planejamento de ações de controle para um robô é necessário definir uma estratégia de planejamento baseada nas características do ambiente e do robô.

Nessa seção é feita uma revisão bibliográfica dos principais métodos de planejamento, com suas vantagens e desvantagens, e uma formulação formal do problema de planejamento de rotas. Além disso, ferramentas utilizadas para o modelamento do problema também são apresentadas.

2.2 Modelamento Geométrico

O primeiro passo para o modelamento geométrico é definir o ambiente em que o robô se encontra. Pode-se definir o ambiente $\mathcal{W}$ do robô como um mundo 2D ou 3D, de tal forma que $\mathcal{W} = \mathbb{R}^2$, ou $\mathcal{W} = \mathbb{R}^3$.

Definimos agora quem se encontra dentro de $\mathcal{W}$ como um robô ou como um obstáculo, dessa maneira, temos:

1. Obstáculos: Partes do ambiente que são permanentemente ocupadas;
2. Robôs: Corpos modelados geometricamente que podem mudar de posição dentro do ambiente a partir de ações de controle.

A região onde se encontram os obstáculos é chamada de $\mathcal{O}$, que é um subconjunto fechado de $\mathcal{W}$, de tal forma que $\mathcal{O} \subseteq \mathcal{W}$.

2.2.1 Modelamento de obstáculos

O conjunto $\mathcal{O}$ pode ser obtido a partir de uma combinação de primitivas H_i, que são subconjuntos de $\mathcal{W}$ facilmente representáveis. Onde um número finito de combinações booleanas, uniões, e interseções podem levar a construção de obstáculos cada vez mais complexos.

Primeiramente consideramos obstáculos convexos em um mundo 2D.

Um subconjunto $X \subset \mathbb{R}^n$ é chamado convexo se, e somente se, para qualquer par de pontos em X, todos os pontos ao longo do segmento de linha que as conecta estão contidos em X. Mais formalmente diz-se que para quaisquer x_1,$x_2 \in X$ e para $\lambda \in [0,1]$,

$$\lambda x_1 + (1-\lambda)x_2 \in X \tag{2.1}$$

Portanto, qualquer interpolação entre x_1e x_2 resulta em pontos dentro de X.

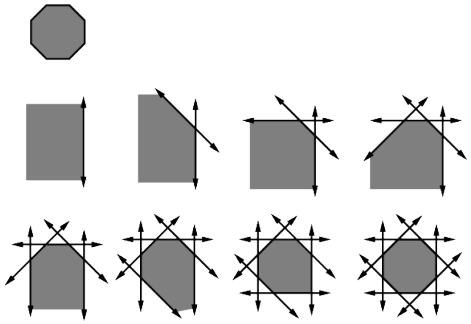


Figura 2.1: Construção de um obstáculo convexo a partir de semi-planos como primitivas geométricas [3]

Uma representação dos limites de $\mathcal{O}$ pode ser um polígono de m lados descrito por vértices ou arestas. Cada vértice corresponde ao canto de um polígono, e cada aresta ao segmento de linha entre os vértices. Uma representação alternativa pode ser obtida usando interseções de m semiplanos como mostrado na figura 2.1.

Esses semiplanos são construídos com base em uma equação de reta $f : \mathbb{R}^2 \rightarrow \mathbb{R}$ tal que $f(x,y) = ax + by + c$. Definimos agora os semiplanos H_i pela inequação:

$$H_i = \{(x,y) \in \mathcal{W} \mid f_i(x,y) \leqslant 0\} \tag{2.2}$$

E agora um obstáculo pode ser definido como:

$$\mathcal{O} = H_1 \bigcap H_2 \bigcap H_3 \bigcap \ldots \bigcap H_m \tag{2.3}$$

Para considerarmos obstáculos não convexos podemos realizar combinações de obstáculos convexos definidos como na equação 2.3:

$$\mathcal{O} = \mathcal{O}_1 \bigcup \mathcal{O}_2 \bigcup \mathcal{O}_3 \bigcup \ldots \bigcup \mathcal{O}_n \tag{2.4}$$

Quando $W = \mathbb{R}^3$, pode-se construir um modelo semelhante de obstáculo. Dessa vez, com poliedros ao invés de polígonos, e semiespaços ao invés de semiplanos.

Seja $f : \mathbb{R}^3 \rightarrow \mathbb{R}$, temos então:

$$f(x, y, z) = ax + by + cz + d \tag{2.5}$$

$$H_i = \{(x, y, z) \in \mathcal{W} \mid f_i(x, y, z) \leq 0\} \tag{2.6}$$

2.2.2 Mapa de Ocupação 3D (*OctoMaps*)

Além de usar primitivas geométricas, pode-se usar uma discretização do mundo em 2D ou 3D, de forma a obter espaços ocupados e desocupados. Uma generalização desse conceito é uma grade de ocupação, onde o valor do espaço, ou célula, não é binário, mas sim um valor num intervalo que pode indicar a probabilidade de um obstáculo existir em uma região de $\mathcal{W}$.

Recentemente, foi apresentada uma forma de apresentar modelos em um ambiente 3D chamada *OctoMap* [4].

Para criar mapas 3D, robôs móveis percebem o ambiente ao tomar medidas de sensores como sonares, sensores de distância a laser, ou outros tipos de sensores de profundidade. Essas medidas possuem incertezas relacionadas à natureza medida, geralmente da ordem de centímetros. Mas também existem incertezas relacionadas a reflexões ou mesmo a obstáculos dinâmicos. Devido a essas incertezas, a tarefa de modelar obstáculos pede uma abordagem probabilística que permite obter resultados mais precisos por meio de técnicas como fusão de sensores.

OctoMaps permitem uma representação probabilística da grade de ocupação além de modelar áreas desocupadas do mapa, tudo isso de uma maneira computacionalmente eficiente. Uma representação eficiente dos dados do mapa é extremamente importante, pois o consumo de memória geralmente é o maior gargalo em sistemas de mapeamento 3D. Para auxiliar nesta realização *OctoMaps* utilizam-se de uma estrutura de dados chamada *Octree.*

2.2.2.1 *Octrees*

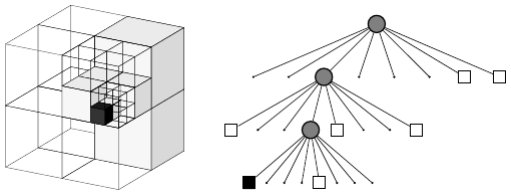
Uma *Octree* é uma estrutura de dados hierárquica para subdivisão espacial em 3D [16]. Cada nó em uma *Octree* representa um voxel, espaço contido em um volume cúbico. Esse volume é subdividido recursivamente em oito sub-volumes até que iguale um volume mínimo que é determinado pela resolução da *Octree* como na figura 2.2 .

Pode-se afirmar que um *OctoMap* é uma *Octree* com valores probabilísticos ao invés de valores binários, ocupado, ou não ocupado.

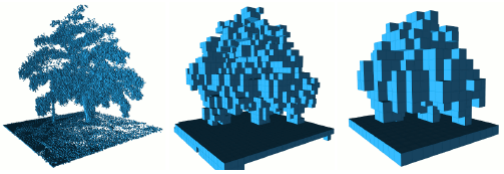
2.2.2.2 Fusão de sensores

Cada nó de um *OctoMap* guarda a probabilidade de ocupação do voxel correspondente. A probabilidade $P(n \mid z_{1:t})$ de um nó n estar sendo ocupado dada uma medida $z_{1:t}$ é estimada por:

$$P(n \mid z_{1:t}) = \left[1 + \frac{1 - P(n \mid z_{1:t})}{P(n \mid z_{1:t})} \frac{1 - P(n \mid z_{1:t-1})}{P(n \mid z_{1:t-1})} \frac{P(n)}{1 - P(n)}\right]^{-1} \tag{2.7}$$



(a) Exemplo de uma *octree* onde os voxels pretos representam células ocupadas e células sombreadas representam células desocupadas. À esquerda, uma representação volumétrica, e à direita uma representação em árvore.



(b) *Octrees* em diferentes resoluções: 0,08m ; 0,64m and 1,28m

Figura 2.2: *Octrees* [4]

Onde z_t é a medida atual, $P(n)$ é a probabilidade anterior, e $P\left(n \mid z_{1:t-1}\right)$ é a estimativa anterior. Assumindo que $P(n) = 0,5$ e usando notação logarítmica, a equação (2.7) pode ser reescrita como:

$$L\left(n \mid z_{1:t}\right) = L\left(n \mid z_{1:t-1}\right) + L\left(n \mid z_t\right) \tag{2.8}$$

Visto que:

$$L(n) = log\left[\frac{P(n)}{1 - P(n)}\right] \tag{2.9}$$

A equação (2.8) permite um cálculo mais rápido da estimativa de ocupação do voxel, pois as multiplicações em (2.7) são substituídas por somas.

Para realização da navegação geralmente é utilizado um limiar para dividir o espaço entre voxels ocupados e não ocupados.

Da equação (2.8) percebe-se que é preciso integrar várias observações no tempo para mudar o estado de um voxel no *OctoMap*. De forma que, se um voxel é determinado livre por k unidades de tempo, ele deverá ser observado como ocupado pela medida (considerando o limiar de probabilidade definido anteriormente) por pelo menos k unidades de tempo. Para que o *OctoMap* possa ser usado em ambientes dinâmicos (que mudam temporariamente ou permanentemente durante o tempo) ao invés de usar a equação (2.8) usamos a equação (2.10):

$$L\left(n \mid z_{1:t}\right) = \left(\left(L\left(n \mid z_{1:t-1}\right) + L\left(n \mid z_t\right), l_{max}\right), l_{min}\right) \tag{2.10}$$

Onde l_{min} e l_{max} são limitadores que diminuem o número de observações necessárias para atualização do estado do voxel.

2.2.2.3 Multi resolução de *OctoMap*

Quando integramos as medidas à *Octree,* somente as folhas da árvore são atualizadas, de modo que aproveitamos a estrutura de árvore para atualizar os demais nós baseado nas folhas.

Assumindo que um volume está ocupado se qualquer parte dele também está, estimamos a probabilidade de ocupação de um nó n a partir dos seus oito nós filhos n_i segundo a seguinte equação:

$$\hat{l}(n) = \max_i \left(L\left(n_i\right) \right) \tag{2.11}$$

Onde $L(n)$ é o logaritmo da probabilidade da ocupação do nó n.

2.2.2.4 Compactação da *OctoMap*

Quando a probabilidade de um dos nós atinge um dos limiares na equação (2.10), o nó é considerado estável. Se todos os filhos de um nó da árvore são estáveis com o mesmo estado de ocupação (livre ou ocupado), então os filhos são retirados da árvore. Se medidas futuras contradizem a estabilidade anterior, os filhos serão regenerados na árvore.

2.3 Transformações de corpo rígido

Todas as técnicas utilizadas para modelar obstáculos em 2.2.1 podem ser utilizadas também para modelar robôs. Vamos agora definir o robô $\mathcal{A}$ como um subconjunto de $\mathbb{R}^2$ ou de $\mathbb{R}^3$, de acordo com a dimensionalidade de $\mathcal{W}$.

Neste trabalho, considera-se que tanto robôs quanto obstáculos são corpos rígidos, de forma que agora serão mostradas maneiras de representar o movimento de $\mathcal{A}$ ou $\mathcal{O}$ em coordenadas.

É necessário definir as coordenadas onde estão sendo feitas as medidas. Todos os pontos do robô ($a \in \mathcal{A}$), estão no sistema de coordenadas do robô. Esse sistema de coordenadas se mexe dentro de um sistema de coordenadas inercial (fixo durante o tempo) onde cada ponto pertence ao espaço $\mathcal{W}$, de maneira que existe uma função h que transforma os pontos do robô em pontos nas coordenadas inerciais em $\mathcal{W}$. Vamos definir a função $h : a \in \mathcal{A} \rightarrow w \in \mathcal{W}$ como:

$$h(\mathcal{A}) = \{h(a) \in \mathcal{W} \mid a \in \mathcal{A}\} \tag{2.12}$$

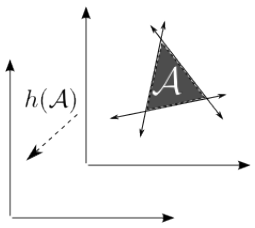


Figura 2.3: Translação do sistema de coordenadas do robô com relação ao sistema de coordenadas inercial

2.3.1 Transformações 2D

Considerando que $\mathcal{A} \subseteq \mathbb{R}^2$ é transladado por dois parâmetros $x_t, y_t \in \mathbb{R}$ temos uma translação definida por:

$$h(x, y) = (x + x_t, y + y_t) \tag{2.13}$$

Para uma rotação anti-horária com um ângulo de $[0, 2\pi)$ em relação à origem das coordenadas do robô, pode-se usar a transformação:

$$(x, y) \mapsto (x.cos(\theta) - y.sen(\theta), x.sen(\theta) + y.cos(\theta)) \tag{2.14}$$

E a matriz de rotação 2x2 :

$$R(\theta) = \begin{pmatrix} cos(\theta) & -sen(\theta) \\ sen(\theta) & cos(\theta) \end{pmatrix} \tag{2.15}$$

De tal forma que:

$$h(x, y) = \begin{pmatrix} x.cos(\theta) - y.sen(\theta) \\ x.sen(\theta) + y.cos(\theta) \end{pmatrix} = R(\theta) \begin{pmatrix} x \\ y \end{pmatrix} \tag{2.16}$$

Para representar uma rotação seguida de uma translação pode-se usar a seguinte matriz homogênea de transformação em 2D no lugar da matriz de rotação da equação (2.15):

$$T = \begin{pmatrix} cos(\theta) & -sen(\theta) & x_t \\ sen(\theta) & cos(\theta) & y_t \\ 0 & 0 & 1 \end{pmatrix} \tag{2.17}$$

2.3.2 Transformações 3D

Uma translação em 3D é uma simples extensão do conceito de translação em 2D. Considerando agora $x_t, y_t, z_t \in \mathbb{R}$ temos a translação definida por:

$$h(x, y, z) = (x + x_t, y + y_t, z + z_t) \tag{2.18}$$

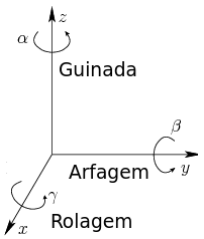


Figura 2.4: Rotações puras em 3D[3]

Em 3D existem três tipos de rotação pura: Guinada, arfagem e rolagem (Fig. 2.4). Qualquer rotação em 3D pode ser representada como uma combinação das três.

As matrizes de rotação pura em 3D são:

$$R_x(\gamma) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos(\gamma) & -sen(\gamma) \\ 0 & sen(\gamma) & cos(\gamma) \end{pmatrix} \tag{2.19}$$

$$R_y(\beta) = \begin{pmatrix} cos(\beta) & 0 & sen(\beta) \\ 0 & 1 & 0 \\ -sen(\beta) & 0 & cos(\beta) \end{pmatrix} \tag{2.20}$$

$$R_z(\alpha) = \begin{pmatrix} cos(\alpha) & -sen(\alpha) & 0 \\ sen(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.21}$$

Três rotações seguidas, rolagem, arfagem e guinada (nessa ordem) podem ser representadas por uma simples multiplicação de matrizes:

$$R(\alpha, \beta, \gamma) = R_z(\alpha)R_y(\beta)R_x(\gamma) \tag{2.22}$$

Pode-se também generalizar a equação homogênea de transformação (rotação + translação) em 2D (2.17) para 3D:

$$T = \begin{pmatrix} R(\alpha, \beta, \gamma) & \cdots & x_t \\ & & y_t \\ \vdots & \ddots & z_t \\ 0 \quad 0 & 0 & 1 \end{pmatrix} \tag{2.23}$$

Usando esse tipo de abordagem é possível chegar em várias matrizes de transformação para um manipulador como o Puma 560 (Fig. 2.5), e então obter as coordenadas do atuador na ponta do manipulador dado um movimento em cada junta do manipulador. Esse cálculo é um problema de cinemática direta. No entanto, ao se planejar o movimento de um manipulador, aparece o problema de cinemática inversa, onde é necessário saber qual a posição de cada junta do robô para que o atuador esteja num certo ponto do espaço. Esse problema é geralmente muito mais difícil

de resolver devido às redundâncias que podem ocorrer em um robô, que levam a infinitas soluções para o problema.

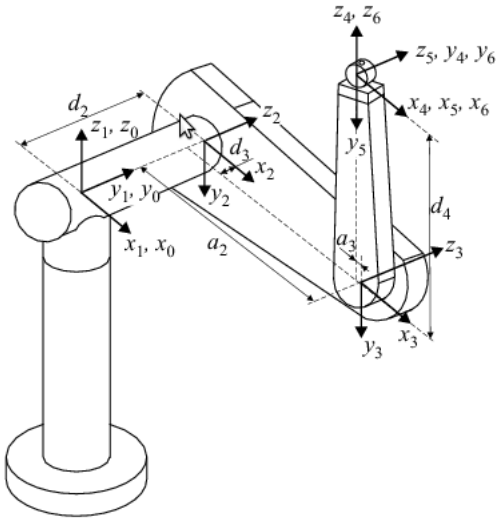


Figura 2.5: Puma 560 e os eixos de coordenadas de cada junta do manipulador. [5]

2.3.3 Quatérnios

Em robótica, é muito comum utilizar quatérnios na representação de rotações em um espaço 3D ($\mathcal{W} = \mathbb{R}^3$). Quatérnios são vetores de quatro dimensões com três componentes pertencentes ao conjunto dos complexos.

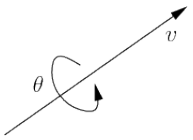


Figura 2.6: Uma rotação em 3D como uma rotação de θ rad em torno de um vetor $\vec{v} = [v_1, v_2, v_3]$.[3]

Seja $\mathbb{H}$ o conjunto dos quatérnios, $h \in \mathbb{H}$ pode então ser representado por $h = a + b\hat{i} + c\hat{j} + d\hat{k}$, sendo que $a, b, c, d \in \mathbb{R}$; e $i, j, k \in \mathbb{C}$ são componentes imaginários do quatérnio, de forma que as seguintes relações são válidas: $i^2 = j^2 = k^2 = ijk = -1$.

Para representar rotações em 3D, usamos um quatérnio unitário, $\|h\| = 1$, ou $a^2+b^2+c^2+d^2 = 1$. Um quatérnio unitário $h = a + bi + cj + dk$ que represente uma rotação de θ rad sobre um vetor em 3D (Fig. 2.6), $\vec{v} = [v_1, v_2, v_3]$, é tal que:

$$h = cos\left(\frac{\theta}{2}\right) + v_1 sen\left(\frac{\theta}{2}\right) i + v_2 sen\left(\frac{\theta}{2}\right) j + v_3 sen\left(\frac{\theta}{2}\right) k \tag{2.24}$$

A partir de (2.24) pode-se obter uma matriz de rotação em 3D semelhante a (2.22):

$$R(h) = \begin{pmatrix} 2(a^2+b^2)-1 & 2(bc-ad) & 2(bd+ac) \\ 2(bc+ad) & 2(a^2+c^2)-1 & 2(cd-ab) \\ 2(bd-ac) & 2(cd+ab) & 2(a^2+d^2)-1 \end{pmatrix} \tag{2.25}$$

No entanto, a representação de uma rotação usando (2.25) não é única. Isso pode ser verificado pela identidade $R(h) = R(-h)$. Ou por uma representação geométrica na figura 2.7 que mostra que uma rotação de θ radianos sobre um vetor $\vec{v}$ é idêntica a uma rotação de $2\pi - \theta$ sobre um vetor $-\vec{v}$.

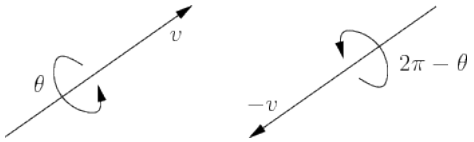


Figura 2.7: Uma mesma rotação representada de maneiras diferentes.[3]

2.4 Topologia

Para facilitar o entendimento de problemas de planejamento de rotas, é útil conhecer algumas definições e conceitos de espaços topológicos.

Um conjunto X é chamado de um espaço topológico se existe uma coleção de subconjuntos de X, que são conjuntos abertos, para os quais valem os seguintes axiomas:

1. A união de um número contável de conjuntos abertos é um conjunto aberto;
2. A interseção de um número finito de conjuntos abertos é um conjunto aberto;
3. X e Ø são conjuntos abertos;

Para o caso em que $X = \mathbb{R}$, conjuntos abertos incluem intervalos abertos do tipo $(0, 1)$ que inclui todos os números reais entre 0 e 1, exceto 0 e 1.

Um conjunto fechado é o complemento de algum conjunto aberto, cujo complemento é um conjunto fechado. O intervalo [0,1], por exemplo, só é fechado porque o seu complemento $(-\infty, 0) \cup (1, +\infty)$ é aberto.

Uma função $f : X \to Y$, onde X e Y são espaços topológicos é contínua se $f^{-1}(O)$ for um conjunto aberto para qualquer conjunto aberto $O \subseteq Y$. Para o qual temos

$$f^{-1}(O) = \{x \in X \mid f(x) \in O\} \tag{2.26}$$

Definida como a pré-imagem de O. Perceba que essa definição não necessita que f tenha inversa.

Agora, suponha uma função bijetiva, $f : X \to Y$ entre espaços topológicos. Se f e f^{-1} são contínuas, então f é um homeomorfismo e X e Y são homeomorfas.

Uma variedade é uma generalização de superfície. Um espaço topológico $M \subseteq \mathbb{R}^m$ é uma variedade se para cada $x \in M$, um conjunto aberto $O \subset M$ existe, tal que,

1. $x \in O$;

2. O é homeomorfa a $\mathbb{R}^n$;

3. n é fixo para todo $x \in M$, onde n é a dimensão da variedade.

Para um espaço topológico X, $(X/\sim)$ é o espaço X redefinido por algum tipo de identificação. Definimos o espaço topológico $\mathbb{S}^1$ um círculo, como [0,1]/ $\sim$, onde a identificação indica que 0 é equivalente a 1. Isso é semelhante a noção de que 0 rad é equivalente a 2π rad. Para qualquer círculo existe um homeomorfismo entre o círculo em questão e $\mathbb{S}^1$.

2.5 Rotas e trajetórias

Definimos agora o conceito de rota. Seja X um espaço topológico, que também é uma variedade. Uma rota será definida uma função contínua, $\tau : [0, 1] \to X$. Uma trajetória define uma rota que não esteja definida somente em X, mas também no tempo, de forma que exista um instante de tempo para cada elemento de τ.

2.6 Espaço de Configurações

O planejamento de rotas para manipuladores e robôs móveis é formalmente realizado através da representação do espaço de configurações.

Suponha um braço robótico que tenha k graus de liberdade. Cada estado, ou configuração do robô, pode ser descrito por k valores reais: $q_1, q_2, \ldots, q_k$ (Fig. 2.8).

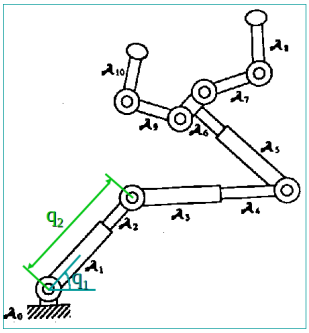


Figura 2.8: Estados de um espaço de configurações.[2]

Esses k valores podem ser abstraídos como um ponto $q = (q_1, \ldots, q_k)$ em um espaço k-dimensional chamado de espaço de configurações $\mathcal{C}$ do robô. Isso permite que seja possível representar formas complexas de um robô como um simples ponto no espaço. Por essa abordagem, é possível compreender várias situações que a princípio parecem muito diferentes devido à sua geometria ou restrições de movimento, como sendo na verdade o mesmo problema de planejamento [1].

Para uma definição mais formal, é necessário o uso do conceito de variedades. O espaço de configurações é um conjunto de transformações que é uma variedade de n dimensões para um robô com n graus de liberdade.

Para resolver um problema de planejamento de rotas, um algoritmo da classe citada na seção 1.3.1.4 faz uma busca no espaço de configurações do robô.

Um robô que permita rotações em dois pontos tem um espaço de configurações assim como na figura 2.9.

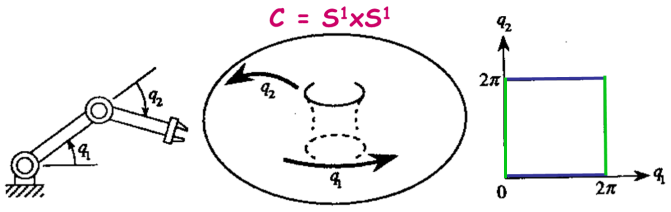


Figura 2.9: Variedade para um certo espaço de configurações.[2]

Além de definir formalmente o espaço de configurações, o uso de conceitos de topologia e suas implicações é muito importante para algoritmos de planejamento de rota.

Tomemos como exemplo um espaço de configurações $\mathcal{C}_1 = \mathbb{R} \times \mathbb{S}^1$, que é basicamente a superfície de um cilindro. Devido a $\mathbb{S}^1$, que possui uma identificação, é possível planejar uma rota que "contorne" o cilindro para chegar a um objetivo. A figura 2.10 mostra uma faixa central que funciona como um obstáculo em $\mathcal{C}_1$, além disso, mostra uma rota possível para alcançar o objetivo q_G a partir de q_I. Esse caminho só é possível por causa da identificação existente em $\mathbb{S}^1$, sem isso, qualquer algoritmo de planejamento de rotas diria ser impossível encontrar uma rota, de forma que pode-se dizer que é muito importante entender a topologia de $\mathcal{C}$ para que potenciais soluções de planejamento não sejam perdidas.

2.6.1 Grupo Especial Euclideano

Dado o grupo de matrizes de rotação n-dimensional $SO(n)$, do qual fazem parte (2.22) e (2.15), temos o grupo especial euclideano $SE(n)$ que corresponde ao conjunto de todas matrizes de transformação de tamanho $(n+1) \times (n+1)$ da forma a seguir:

$$\left\{ \begin{pmatrix} R & v \\ 0 & 1 \end{pmatrix} \mid R \in SO(n),\, v \in \mathbb{R}^n \right\} \tag{2.27}$$

É possível perceber que (2.27) é uma generalização das matrizes de transformação homogênea

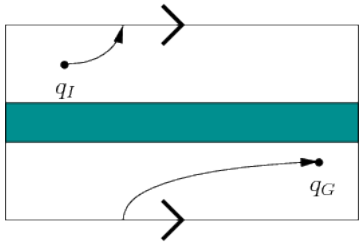


Figura 2.10: Rota "contornando" uma superfície cilíndrica para chegar ao objetivo q_G a partir de q_I.[3]

(2.17) e (2.23), que levam um objeto a transladar e rotacionar em $\mathcal{W}$.

Pode-se verificar que o grupo especial euclideano $SE(2)$ é homeomorfo a $\mathbb{R}^2 \times SO(2)$, e que $SE(3)$ é homeomorfo a $\mathbb{R}^3 \times SO(3)$ [3].

Para um robô que se movimenta no plano, o espaço de configurações é então $SE(2)$, pois o robô pode se movimentar transladando e rotacionando com relação ao próprio eixo, ou seja, possui três graus de liberdade. $SE(2)$ é a variedade usada para planejamento de corpos rígidos em 2D.

Para planejamento de corpos rígidos em 3D, a variedade usada geralmente é $SE(3)$ pois geralmente o corpo pode se movimentar livremente no espaço transladando em $\mathbb{R}^3$ e rotacionando em torno do próprio eixo conforme $SO(3)$.

2.6.2 Revisitando Quatérnios

Seja um quatérnio unitário como definido na seção 2.3.3, usando as propriedades de multiplicação entre os componentes imaginários dos quatérnios pode-se chegar a uma multiplicação entre quatérnios $h_3 = h_1.h_2$ de forma que os elementos de h_3 são:

$$\begin{aligned} a_3 &= a_1a_2 - b_1b_2 - c_1c_2 - d_1d_2 \\ b_3 &= a_1b_2 + a_2b_1 + c_1d_2 - c_2d_1 \\ c_3 &= a_1c_2 + a_1c_2 + b_2d_1 - b_1d_2 \\ d_3 &= a_1d_2 + a_2d_1 + b_1c_2 - b_2c_1 \end{aligned} \tag{2.28}$$

Usando essa operação é possível mostrar que $\mathbb{H}$ é um grupo com respeito a multiplicação de quatérnios. Além disso é possível mostrar que o grupo de quatérnios unitários é isomorfo ao grupo de matrizes de rotação 3-dimensional $SO(3)$ [3], isto é, é possível realizar rotações no grupo dos quatérnios unitários e mapear de volta os resultados para $SO(3)$ sem prejuízo ao resultado final.

Para rotacionar um ponto $(x, y, z) \in \mathbb{R}^3$ usando um quatérnio unitário $h \in \mathbb{H}$, onde $h = a + bi + cj + dk$, usamos a seguinte transformação:

$$p' = h.p.\overset{*}{h} \tag{2.29}$$

Onde $p, \overset{*}{h} \in \mathbb{H}$,e $p = 0 + xi + yj + zk$, e $\overset{*}{h}$ é o quatérnio conjugado de h, definido por $\overset{*}{h} = a - bi - cj - dk$.

Seja agora

$$p' = w + x'i + y'j + z'k \tag{2.30}$$

Então temos que o ponto rotacionado será dado por

$$h(x, y, z) = (x', y', z') \tag{2.31}$$

Para realizar a rotação, são necessárias multiplicações entre quatérnios. Verifica-se da definição de multiplicação entre quatérnios (2.28) que ela é não comutativa, assim como as rotações em 3D também não são.

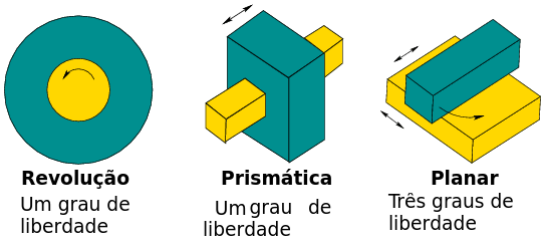


Figura 2.11: Três tipos de juntas entre corpos rígidos

A grande vantagem de usar quatérnios, e o porquê dele ser muito usado em aplicações computacionais, é que é mais compacto guardar quatro valores para um quatérnio do que nove valores para uma matriz de rotação, além de que para rotacionar um ponto em 3D são realizadas menos operações usando aritmética de quatérnios do que usando matrizes de rotação.

2.6.3 Combinações de configurações de espaço

Se existirem n múltiplos corpos rígidos que se movem independentemente, então pode ser usado o produto cartesiano entre cada espaço de configuração individual para obter um único espaço de configurações:

$$\mathcal{C} = \mathcal{C}_1 \times \mathcal{C}_2 \times \mathcal{C}_3 \times \mathcal{C}_4 \times \ldots \times \mathcal{C}_n \tag{2.32}$$

No entanto, se esses mesmos corpos rígidos forem anexados um ao outro, então o espaço de configuração do corpo resultante deve ser analisado caso a caso.

Um bom exemplo de como a análise caso a caso é sempre necessária é uma junta de revolução como na figura 2.11 . De modo geral, ela não possui limites de rotação, de forma que o espaço de configuração associado a ela é homeomorfo a $SO(2)$. Entretanto, se houver limites de rotação para a junta, o espaço de configurações associado é homeomorfo a $\mathbb{R}$.

2.6.4 Obstáculos no espaço de configurações

Um algoritmo de planejamento deve levar em conta os obstáculos dentro de $\mathcal{W}$ que estejam em regiões alcançáveis de $\mathcal{C}$. Para isso, primeiro é retirado do espaço de configurações uma região de obstáculos, onde é garantido que o robô $\mathcal{A}$ irá colidir consigo mesmo ou com algum obstáculo $\mathcal{O}$.

Partindo para uma definição formal, suponha uma região de obstáculos $\mathcal{O} \subset \mathcal{W}$. Assuma o robô rígido $\mathcal{A} \subset \mathcal{W}$.

Considere que robô $\mathcal{A}$ é modelado como uma coleção de m corpos rígidos $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \cdots, \mathcal{A}_m$ que podem ou não estar ligados por meio de juntas. Um único ponto de configuração q é dado para toda a coleção de corpos rígidos (Fig. 2.8). Seja P um conjunto de pares de colisão no qual cada par $(i, j) \in P$ representa uma colisão entre $\mathcal{A}_i$ e $\mathcal{A}_j$.

A região de obstáculos mencionada anteriormente, $\mathcal{C}_{obs} \subset \mathcal{C}$, é definida como:

$$\mathcal{C}_{obs} = \left(\bigcup_{i=1}^{m} \{q \in \mathcal{C} \mid \mathcal{A}_i(q) \cap \mathcal{O} \neq \emptyset\} \right) \bigcup \left(\bigcup_{[i,j] \in P} \{q \in \mathcal{C} \mid \mathcal{A}_i(q) \cap \mathcal{A}_j(q) \neq \emptyset\} \right) \tag{2.33}$$

Ou seja, uma configuração $q \in \mathcal{C}$ está na região de obstáculos se uma parte do robô colide com $\mathcal{O}$, ou com outra parte do robô.

Todas as configurações que não estão na região de obstáculos são definidas pelo espaço de configurações livre:

$$\mathcal{C}_{free} = \mathcal{C} \setminus \mathcal{C}_{obs} \tag{2.34}$$

2.6.5 Definição do problema básico de manipulação de rotas

Dado um ambiente $\mathcal{W} = \mathbb{R}^2$ ou $\mathcal{W} = \mathbb{R}^3$, uma região de obstáculos $\mathcal{O} \subset \mathcal{W}$, e um robô de corpo rígido $\mathcal{A}$, pode-se então determinar um espaço de configurações $\mathcal{C}$ que especifica o conjunto de todas as possíveis transformações que podem ser apicadas ao robô $\mathcal{A}$. De $\mathcal{C}$, obtém-se $\mathcal{C}_{free}$ e $\mathcal{C}_{obs}$.

Feito isso, dadas as configurações inicial e final $q_I, q_G \in \mathcal{C}_{free}$, desde que exista pelo menos uma rota, um algoritmo de planejamento de rotas completo deve calcular uma rota contínua $\tau : [0,1] \to \mathcal{C}_{free}$, tal que $\tau(0) = q_I$ e $\tau(1) = q_G$. Se não houver solução o algoritmo completo deve identificar que não existe rota que resolva o problema.

O problema é resumido graficamente na figura 2.12 .

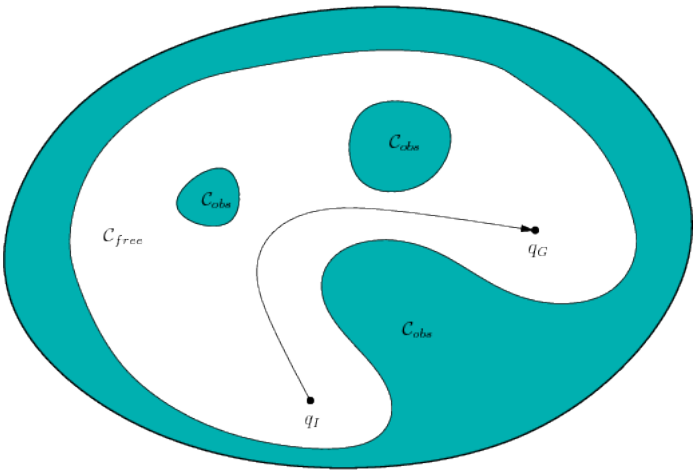


Figura 2.12: Visualização do problema básico de manipulação de rotas, onde um algoritmo leva o robô de uma configuração inicial até uma configuração final por meio de uma rota livre de obstáculos.[3]

2.7 Espaços Métricos

Em muitos algoritmos de planejamento de rotas é necessário prover uma maneira de se medir quantitativamente uma distância entre dois pontos em um mesmo espaço de configurações. Para que isso seja feito, é preciso definir o que é um espaço métrico.

Um espaço métrico (X, ρ) é um espaço topológico X que possui uma função $\rho : X \times X \to \mathbb{R}$, tal que, para quaisquer $a, b, c \in X$ valem os seguintes axiomas [3]:

1. $\rho(a, b) \geq 0$;
2. $\rho(a, b) = 0 \longleftrightarrow a = b$;
3. $\rho(a, b) = \rho(b, a)$;
4. $\rho(a, b) + \rho(b, c) \geq \rho(a, c)$;

2.8 Direção diferencial

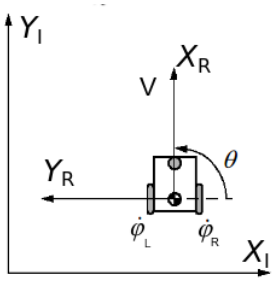


Figura 2.13: Robô com direção diferencial mostrado com seu referencial inercial e seu referencial local

Robôs com direção diferencial possuem um eixo de coordenadas local com origem no ponto médio entre os eixos de rotação de cada roda, como visto na figura 2.13. A figura 2.13 também mostra o eixo de coordenadas inercial, e como o robô está rotacionado do eixo de coordenadas local para o eixo inercial.

O controle que temos sobre o robô resume-se às velocidades angulares dos eixos de cada roda $\dot{\varphi}_R$ e $\dot{\varphi}_L$(Fig. 2.14). O que deseja-se é primeiro construir um modelo cinemático que use essas velocidades como ações de controle para gerar uma mudança de velocidade no robô, e por último que essa velocidade seja mapeada para o referencial inercial.

O robô possui uma distância entre as rodas, l, e o raio das rodas é r.

Devido às rodas fixas, a velocidade do robô no eixo Y_R será nula, ou seja $\dot{y}_R = 0$, enquanto que a velocidade no eixo X_R é dada pela contribuição individual de cada roda.

Se uma roda tem velocidade nula, devido ao fato de que a origem do eixo está no ponto médio entre os eixos das rodas, a outra roda contribuirá com metade da velocidade, portanto $\dot{x}_R = \frac{r.\dot{\varphi}_R}{2} + \frac{r.\dot{\varphi}_L}{2}$.

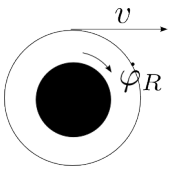


Figura 2.14: Velocidades angular e linear nas rodas do robô diferencial

Visto que a velocidade linear do robô nas coordenadas $\{X_R, Y_R\}$ será uma função dependente somente do valor de $\dot{x}_R$, vamos daqui para frente representá-la como um escalar v, ao invés da representação usual para vetores.

Agora que temos a velocidade linear do robô, precisamos da velocidade angular $\dot{\theta}$. Assim como foi feito para a velocidade linear, calculamos a contribuição de cada roda primeiro.

Se uma das rodas está fixa, a velocidade da outra roda vai fazer o robô girar em torno da roda fixa. Como uma velocidade angular positiva da roda direita gera um aumento de θ, a contribuição dela será positiva, o contrário acontece para a roda esquerda.

Portanto, temos o modelo cinemático para o eixo de coordenadas do robô como sendo

$$\dot{\xi_R} = \begin{pmatrix} \dot{x}_R \\ \dot{y}_R \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} \frac{r.\dot{\varphi}_R}{2} + \frac{r.\dot{\varphi}_L}{2} \\ 0 \\ \frac{r.\dot{\varphi}_R}{2l} - \frac{r.\dot{\varphi}_L}{2l} \end{pmatrix} = \begin{pmatrix} v \\ 0 \\ \dot{\theta} \end{pmatrix} \tag{2.35}$$

Para obtermos o modelo cinemático para o eixo de coordenadas inercial, observamos que

$$\dot{\xi_R} = R(\theta)\dot{\xi}_I \tag{2.36}$$

Onde $R(\theta)$ é a matriz de rotação (2.15). Então temos que

$$\dot{\xi}_I = R^{-1}(\theta)\dot{\xi}_R = \begin{pmatrix} \dot{x}_I \\ \dot{y}_I \\ \dot{\theta} \end{pmatrix} \tag{2.37}$$

$$\dot{\xi}_I = \begin{pmatrix} \frac{r}{2}\left(\dot{\varphi}_R + \dot{\varphi}_L\right) cos(\theta) \\ \frac{r}{2}\left(\dot{\varphi}_R + \dot{\varphi}_L\right) sen(\theta) \\ \frac{r}{2l}\left(\dot{\varphi}_R - \dot{\varphi}_L\right) \end{pmatrix} = \begin{pmatrix} v\, cos(\theta) \\ v\, sen(\theta) \\ \dot{\theta} \end{pmatrix} \tag{2.38}$$

Observa-se que o modelo cinemático pode ser expressado tão somente pela velocidade linear $v\left(\dot{\varphi}_R, \dot{\varphi}_L\right)$ e pela velocidade angular $\dot{\theta}\left(\dot{\varphi}_R, \dot{\varphi}_L\right)$. Essa representação é útil porque alguns sistemas oferecem ao usuário uma interface para as velocidades lineares e angulares ao invés da velocidade das rodas.

2.9 Holonomicidade

O conceito de holonomicidade é usado para caracterizar restrições cinemáticas associadas ao movimento de uma base móvel robótica.

Um robô é dito não-holonômico se possuir ao menos uma restrição não-holonômica. Onde uma restrição cinemática holonômica, é uma restrição que pode ser expressada como uma função somente das variáveis de posição [1]. Se um robô possuir uma restrição cinemática que dependa de alguma derivada no tempo das variáveis de posição então ele é não-holonômico.

Também é possível identificar mais facilmente a holonomicidade de um robô pelo número de graus de liberdade e o número de ações de controle do mesmo. Isso porque é possível dizer que uma base móvel robótica é holonômica se, e somente se, o seu número de graus de liberdade (dimensionalidade do seu espaço de configurações) for igual ao seu número de variáveis de ação de controle.

Um robô com direção diferencial como descrito na seção 2.8 possui duas ações de controle, $(\dot{\varphi}_R, \dot{\varphi}_L)$ ou $\left(v, \dot{\theta}\right)$, e três graus de liberdade, portanto todos robôs com direção diferencial são não-holonômicos.

2.10 Métodos numéricos para resolução de equações diferenciais

Muitos fenômenos físicos podem ser representados através de equações diferenciais. O modelamento cinemático e/ou dinâmico de restrições associadas a robôs geralmente é feito dessa maneira. E é muito comum que não haja uma resposta fechada para tais equações diferenciais, por isso geralmente são obtidas soluções através de métodos numéricos.

São apresentados agora dois métodos muito usados para resolver equações diferenciais, o método de Euler, e o método de Runge Kutta de ordem 4.

2.10.1 Método de Euler

Seja uma equação diferencial de primeira ordem

$$\frac{dy}{dt} = f(t, y) \tag{2.39}$$

com valor de contorno

$$y\left(t_0\right) = y_o \tag{2.40}$$

Vamos supor que existe uma solução para essa equação diferencial em um intervalo de interesse.

Dado um passo $h \in \mathbb{R}$, vamos aproximar um valor $y\left(t_0 + h\right)$ pelo valor da reta tangente em $t_o + h$. Podemos repetir esse processo quantas vezes preciso para obter valores de y:

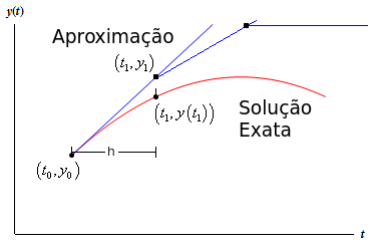


Figura 2.15: Método de Euler com passo h

$$y_{n+1} = y_n + f\left(t_n, y_n\right) h \tag{2.41}$$

onde $y_{n+1} = y\left(t_{n+1}\right)$, e $t_{n+1} = t_n + h$.

Dessa maneira obtém-se uma sequência de valores $y_1, y_2, \ldots, y_n$ que aproximam os valores da solução exata nos pontos t_1,$t_2, \ldots, t_n$ [17]. A figura 2.15 mostra uma interpretação visual para o método de Euler.

O método de Euler possui um erro de truncamento (diferença entre valor exato e valor aproximado) proporcional a h^2, portanto ele precisa de valores muitos pequenos de h para encontrar uma solução satisfatória. O problema de diminuir o passo indiscriminadamente é que assim aumenta-se grandemente o número de iterações necessárias para encontrar uma solução.

2.10.2 RK-4

O método de Runge-Kutta de quarta ordem, ou RK-4, é muito parecido com o método de Euler, no entanto ele usa uma fórmula diferente para aproximar os valores.

$$y_{n+1} = y_n + \left(k_1 + 2k_2 + 2k_3 + k_4\right) \frac{h}{6} \tag{2.42}$$

Sendo que

$$\begin{gathered} k_1 = f\left(t_n\right) \\ k2 = k3 = f\left(t_n + \tfrac{h}{2}\right) \\ k_4 = f\left(t_n + h\right) \end{gathered} \tag{2.43}$$

O RK-4 possui um erro de truncamento proporcional a h^5 [17], sendo portanto um método mais preciso do que o método de Euler. No entanto, devido ao maior número de fórmulas ela tem um custo computacional maior do que o de Euler, se $f(t, y)$ não for muito complicada, ou não forem necessárias muitas iterações para chegar ao valor desejado, isso não constitui um problema [17].

2.10.3 Sistemas de equações diferenciais

Os métodos das seções anteriores também podem ser usados para resolver um sistema de equações diferenciais de primeira ordem .

Seja o problema de valor inicial dado por:

$$\frac{\partial \vec{x}}{\partial t} = f(t, \vec{x}); \qquad \vec{x}\,(t_0) = x_0 \tag{2.44}$$

O método de Euler é generalizado para:

$$\vec{x}_{n+1} = \vec{x}_n + hf(t_n, \vec{x}_n) \tag{2.45}$$

RK-4 é generalizado para:

$$\vec{x}_{n+1} = \vec{x}_n + \left(\vec{k}_{n1} + 2\vec{k}_{n2} + 2\vec{k}_{n3} + \vec{k}_{n4}\right)\frac{h}{6} \tag{2.46}$$

Sendo que

$$\begin{aligned} \vec{k}_{n1} &= f\left(t_n, \vec{x}_n\right) \\ \vec{k}_{n2} &= f\left(t_n + \tfrac{h}{2}, \vec{x}_n + \tfrac{h}{2}\vec{k}_{n1}\right) \\ \vec{k}_{n3} &= f\left(t_n + \tfrac{h}{2}, \vec{x}_n + \tfrac{h}{2}\vec{k}_{n2}\right) \\ \vec{k}_{n4} &= f\left(t_n + h, \vec{x}_n + h\vec{k}_{n3}\right) \end{aligned} \tag{2.47}$$

Todas as propriedades das seções anteriores também valem para a generalização de problemas com sistemas de equações diferenciais.

2.11 Planejamento baseado em amostragem do espaço de configurações

Na prática, a construção de $\mathcal{C}_{obs}$ é bastante complicada devido a complexidade e o número de obstáculos que aparecem na vida real. Construir e manter uma caracterização geométrica dos obstáculos de um robô ao longo do tempo não é uma tarefa tão complicada se *OctoMaps* forem usados, por exemplo, mas os algoritmos de planejamento das seções 1.3.1.3-1.3.1.1 calculam uma rota baseados no conhecimento de todos os obstáculos existentes, o que pode levar a um planejamento muito lento, principalmente se for considerado um ambiente dinâmico (com obstáculos se movendo).

Os algoritmos de planejamento baseados em amostragem utilizam uma abordagem para lidar com obstáculos que é muito mais eficiente para a maioria dos casos pois ela somente se preocupa com os obstáculos quando realmente é necessário.

A conexão entre a geometria do robô e o espaço de configurações livre é feita por um detector de colisões. Esta abstração permite que o planejador de rotas seja independente do modelo geométrico utilizado para representar o robô, assim como da representação dos obstáculos. Assim,

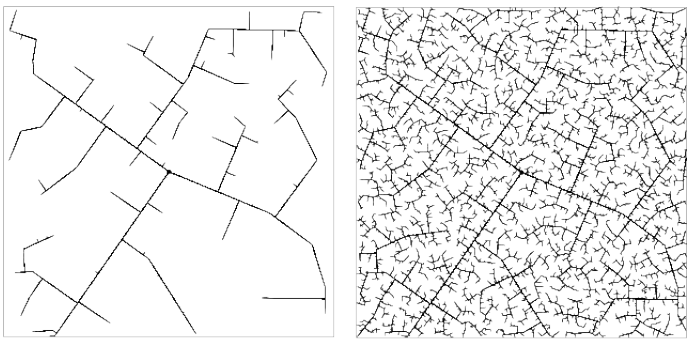


Figura 2.16: Amostragem de um espaço de configurações usando o algoritmo RRT. Ele tende a amostrar todo o espaço de configurações conforme aumenta o número de iterações [3].

o planejador tem a função de gerar amostras no espaço de configurações, enquanto o detector de colisões tem a função de verificar se elas devem ser rejeitadas ou aceitas. Essa é a grande vantagem dos planejadores baseados na amostragem do espaço de configurações, pois $\mathcal{C}_{free}$ não precisa ser calculado explicitamente. Dessa maneira, os algoritmos de planejamento funcionam amostrando e procurando $\mathcal{C}$ incrementalmente por um caminho, gradualmente revelando uma rota plausível através do detector de colisões [12, 6].

A maioria dos algoritmos dessa classe são probabilisticamente completos, isto é, com pontos suficientes de amostragem, a probabilidade de que o algoritmo ache uma solução, se ela existir, é de 100% (Fig. 2.16). É possível no entanto que a solução não exista, ou que a amostragem de muitos pontos demore um tempo impraticável para aplicações em tempo real para encontrar a solução. Portanto, o algoritmo que leva muito tempo para encontrar uma solução pode se encontrar nesses dois casos.

Essa classe de algoritmos possui muitas semelhanças entre si, inclusive é possível delinear um *framework* básico para os seus algoritmos :

Seja $\mathcal{G}(V, E)$ um grafo de busca não-direcionado, onde cada vértice é uma configuração do robô em $\mathcal{C}_{free}$, e cada aresta é uma rota que conecta duas configurações então é possível definir um *framework* básico para essa classe de algoritmos.

1. Inicialização: Inicialize $\mathcal{G}(V, E)$ com a configuração inicial e final do grafo, respectivamente q_I, q_G ;
2. Método de seleção de vértice (MSV): Selecione um vértice $q_{cur} \in V$ para expansão do grafo;
3. Método de planejamento local (MPL): Selecione um vértice q_{new} que pode ou não estar no grafo, neste momento é feita a detecção de colisões e verificação se existe um caminho possível entre q_{cur} e q_{new}, se sim, adicione q_{new} a $\mathcal{G}$, senão, volte ao passo 2;
4. Solução: Cheque se o grafo resolve o problema de questionamento único alcançando q_G, se isso acontecer uma solução foi encontrada, senão volte ao passo 2.

Perceba que essa classe de algoritmos não é capaz de saber por si só que não é possível encontrar uma solução quando a solução não existe. Isso deve ser feito pelo usuário com algum tipo de

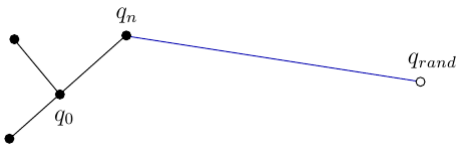


Figura 2.17: Construção de uma RRT.[6]

Algorithm 2.1 RRT geométrico

RRT(q_I)

```
1 G.inicializa(q_I);
2         repeat
3                 q_rand=RANDOM_CONFIG(c-space);
4                 q_near=NEAREST(G,q_rand);
5                 G.adiciona_aresta(q_near,q_rand);
```

heurística própria, como número máximo de iterações, ou tempo máximo de execução do algoritmo.

Os métodos de planejamentos baseados em árvores começam amostrando aleatoriamente um nó no espaço de configurações livres a partir de uma configuração inicial. Então o planejador aplica uma heurística de expansão, que geralmente dá ao método o seu nome, da qual a amostra é conectada a árvore.

Algoritmos dessa classe ainda podem ser divididos em algoritmos que planejam geometricamente, baseados na crença de que o robô não possui nenhuma restrição cinemática ou dinâmica (também chamado de restrição cinedinâmica) para ir de uma configuração livre para outra; e algoritmos que planejam rotas baseados em restrições cinedinâmicas.

A fim de ilustrar esse *framework* básico são mostrados dois algoritmos bastante usados para planejamento de rotas, o RRT, e o RRT-Connect.

2.11.1 Planejamento Geométrico

2.11.1.1 RRT's

Árvores aleatórias para exploração rápida, ou mais comumente chamado pela sigla em inglês RRT, exploram agressivamente o espaço de configurações livres de um robô, expandindo uma árvore incrementalmente a partir da configuração inicial q_I. A versão original do RRT é usada para um planejamento cinedinâmico, aqui é mostrada uma versão mais simples, puramente geométrica (algoritmo 2.1) que serve para expandir a árvore assim como na figura 2.17.

O primeiro passo inicializa o grafo, com um vértice para a configuração inicial, e sem arestas; no terceiro passo uma configuração aleatória é obtida dentro do espaço de configurações (não no espaço de configurações livres, pois aqui ele não necessita ser explicitado!); no quarto passo ele acha o ponto do grafo mais próximo ao q_{rand} de acordo com alguma métrica como foi mencionado na seção 2.7, e adiciona ele à árvore.

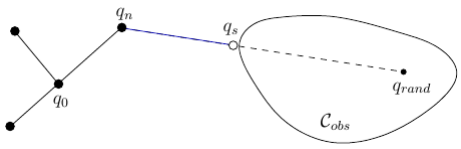


Figura 2.18: Se existe um obstáculo, a aresta se extende até os limites do obstáculo tanto quanto o detector de colisões permite.[6]

Algorithm 2.2 RRT geométrico com obstáculos

RRT(q_I):

```
1
2  G.inicializa(q_I);
3          repeat
4                  q_n=NEAREST(G,q_rand);
5                  q_s=STOPPING_CONFIGURATION(q_n,q_rand);
6                  if (q_s != q_n) then
7                          G.adiciona_vertice(q_s);
8                          G.adiciona_aresta(q_n,q_s);
```

Essa abordagem não leva em conta obstáculos. Para planejar para obstáculos o algoritmo 2.2 utiliza o procedimento STOPPING_CONFIGURATION para achar a configuração mais próxima possível de q_{rand}, um q_s (Fig. 2.18)

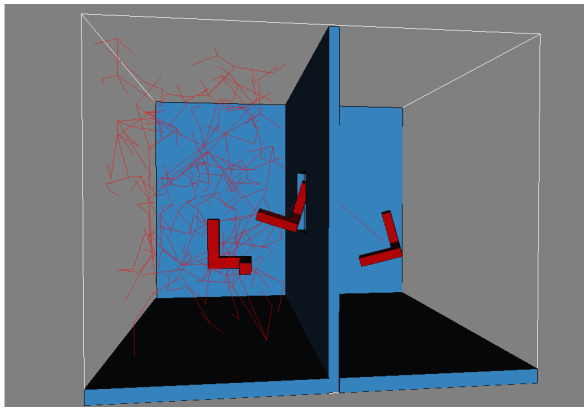
Para usar uma RRT para produzir um τ, como definido na seção 2.6.5, usamos um parâmetro $0 < goal_{bias} < 1$, onde $goal_{bias} = 0,05$ significa que a cada iteração o algoritmo tem uma chance de 5% de tentar conectar a árvore a configuração final. Valores tipicamente usados de $goal_{bias}$variam de 1% a 5%.

2.11.1.2 RRT-Connect

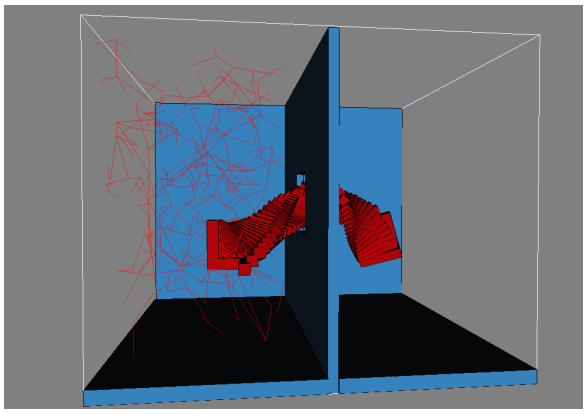
O algoritmo RRT-Connect é basicamente o mesmo algoritmo que o RRT sem restrições cinedinâmicas com a diferença básica de que ele usa duas árvores RRT ao invés de uma. Uma dessas árvores é chamada de $\mathcal{T}_a$ e cresce a partir de q_I, e a outra árvore, $\mathcal{T}_b$, cresce a partir de q_G, por essa razão também é chamado de RRT-bidirecional [3, 18].

Como visto no algoritmo 2.3, após a inicialização, o RRT-connect expande a árvore da configuração inicial $\mathcal{T}_a$ como no algoritmo 2.2, então ele tenta conectar a árvore da configuração final $\mathcal{T}_b$ a última expansão da árvore da configuração final, se isso não for possível, ele expande $\mathcal{T}_b$ até o ponto mais próximo possível. Se for possível conectar as duas árvores, então a solução foi encontrada, caso contrário, as duas árvores trocam de função, a árvore que estava expandindo agora vai tentar se conectar, e a que estava tentando se conectar tentará se expandir.

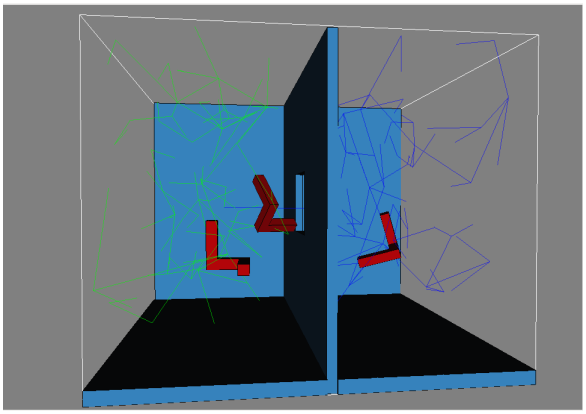
Devido a essa característica de procurar rapidamente uma solução, o algoritmo RRT-connect dificilmente vai explorar todo o espaço de configurações como na figura 2.16, no entanto, convergirá para uma solução muito mais rapidamente que o RRT simples (figura 2.19).



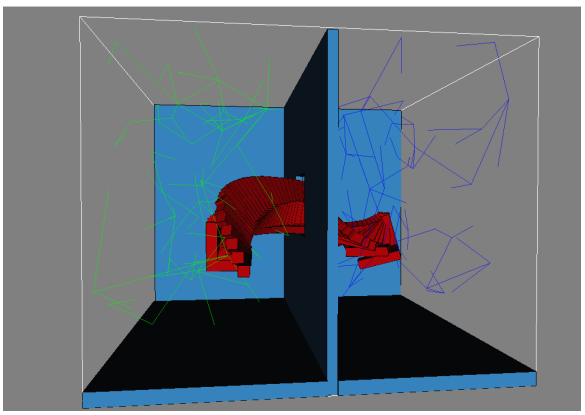
(a) RRT. Corpo rígido se movendo na rota encontrada



(b) RRT. Rota encontrada



(c) RRT-connect. Corpo rígido se movendo na rota encontrada.



(d) RRT-Connect. Rota encontrada.

Figura 2.19: Comparação entre RRT e RRT-Connect. Para o RRT-Connect arestas verdes mostram a árvore partindo da configuração inicial, e arestas azuis mostram a árvore que parte da configuração final. Para ambos os casos temos $goal_{bias} = 5\%$.

Algorithm 2.3 RRT-connect, ou RRT-bidirecional

RRT_CONNECT(q_I,q_G):

```
1  T_a.inicializa(q_I);
2  T_b.inicializa(q_G);
3  repeat
4          //EXPANDE
5          q_n=NEAREST(T_a,q_rand);
6          q_s=STOPPING_CONFIGURATION(q_n,q_rand);
7          if q_s != q_n then
8                  T_a.adiciona_vertice(q_s);
9                  T_a.adiciona_aresta(q_n,q_s);
10
11                 //CONECTA
12                 q_n2=NEAREST(T_b,q_s);
13                 q_s2=STOPPING_CONFIGURATION(q_n2,q_s)
14                 if q_s2 != q_n2 then
15                         T_b.adiciona_vertice(q_s2);
16                         T_b.adiciona_aresta(q_n2,q_s2);
17                 if q_s2 == q_s then return SOLUTION;
18         SWAP(T_a,T_b);
```

2.11.2 Planejamento com restrições diferenciais

Em alguns espaços de configuração o método de planejamento local (MPL) possui restrições sobre as quais planejamentos locais são válidos ou não. Para um carro com direção *Ackermann* não é possível se mover lateralmente por exemplo, isso é devido ao seu modelo cinemático que possui restrições de movimento.

Para resolvermos problemas de planejamento de rotas desse tipo precisamos reformular o problema de planejamento de rotas definido na seção 2.6.5 de forma que possamos incluir uma forma de prever movimentos plausíveis pelo modelo cinemático.

Seja X um estado de espaços com a seguinte equação de transição $\dot{x} = f(x, u)$ e um espaço, que contém todas as ações de controle do robô, U. Dado um estado inicial $x_I \in X$ e uma região próxima à configuração final $X_G \subset X$, a tarefa do planejador fica sendo a de calcular uma função $u : [0, t] \to U$ que corresponde a uma trajetória $q : [0, t] \to X_{free}$, onde $q(0) = x_I$ e $q(1) = x_G$.

2.11.2.1 RRT

O algoritmo original para RRT, em linhas gerais, possui poucos passos a mais do que o do algoritmo 2.1.

RRT continua amostrando probabilisticamente uma configuração (x_{rand}), e agora também amostra uma ação de controle $u \in U$, que tenta chegar até a configuração amostrada anteriormente,

através de uma propagação do modelo dinâmico do sistema de uma configuração x_n para uma configuração x_r próxima a x_{rand}.

Para realizar essa propagação o algoritmo deve integrar a equação diferencial da restrição cinedinâmica. Se a restrição for cinemática as equações diferenciais do sistema serão de primeira ordem, de forma que nesse caso o algoritmo pode usar as equações (2.45), (2.46) para aproximar uma propagação do modelo cinemático de uma direção diferencial. Essa tarefa de amostrar no espaço de controle e de resolver numericamente o sistema que é chamada de MPL no algoritmo 2.4 .

É importante notar que a propagação do modelo dinâmico é possível para "avançar" com o sistema, no entanto não é possível fazer uma abordagem como a do algoritmo 2.3, pois a volta do sistema para o passado não é bem definida, impossibilitando que a árvore da configuração final se expanda, ou se conecte à configuração inicial.

Algorithm 2.4 RRT (para planejamento com restrições cinedinâmicas)

RRT(q_I):

```
1 G.inicializa(q_I);
2         repeat
3                 x_rand=RANDOM_CONFIG(c-space);
4                 x_near=NEAREST(G,x_rand);
5                 (u,x_r)=MPL(x_n,x_rand)
6                 G.adiciona_aresta(x_near,x_r);
```

Capítulo 3

Desenvolvimento

3.1 *Pioneer*

Pioneer é uma família de robôs móveis com duas ou quatro rodas (Fig. 3.1), que possuem direção diferencial como visto na seção 2.8.

Dentre suas várias características destacam-se um *array* de sonares com oito sonares, até 4 motores DC para direção diferencial, e um computador embarcado no robô.

Todos eles podem ser diretamente controlados através de um *joystick* conectado por um cabo pela parte traseira do robô como na figura 3.2. Além disso, como medida de segurança, o *pioneer* possui um botão de parada de emergência que desliga a fonte de energia dos motores aos controladores dos motores DC.

3.2 Descrição do Porthos

No laboratório de robótica e automação (LARA), existem três robôs da família *pioneer*, desses três existem dois robôs modelo P3-AT, e um modelo P3-DX. O robô usado para esse trabalho é comumente chamado de Porthos (Fig. 3.3) pelo grupo de pesquisa em robótica móvel, o AMORA (Autonomous MObile Robot Algorithms), e aqui também será denominado dessa maneira.

Figura 3.1: Robô da família *Pioneer*

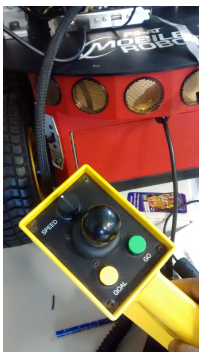


Figura 3.2: Joystick para comandar um *pioneer* diretamente

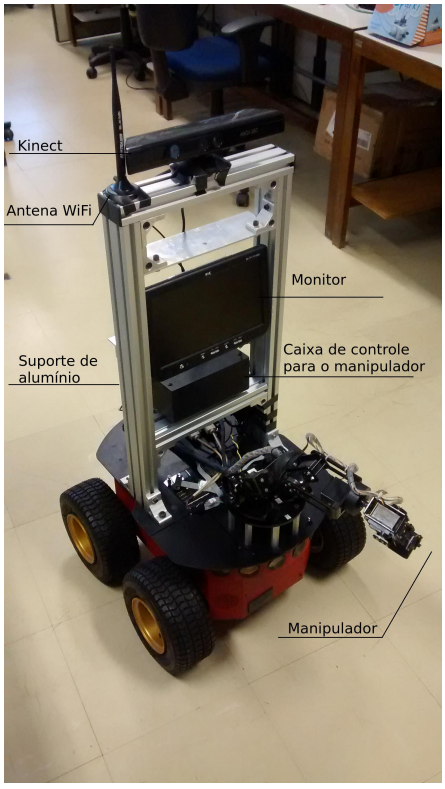


Figura 3.3: *Porthos* - *Pioneer* modificado para manipulação móvel

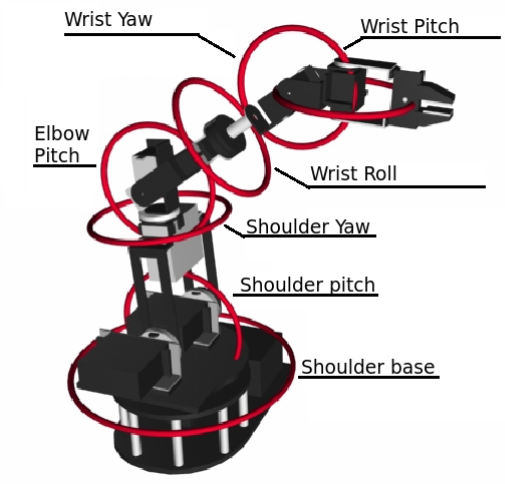


Figura 3.4: Ilustração do *Cyton* I e suas respectivas juntas

O Porthos possui além de direção diferencial e as demais características de um robô *pioneer*, um computador embarcado com processador *Pentium M* de 2,0GHZ e 512Mb de memória, um *kinect*, uma caixa de som, um televisor/monitor de 11", um braço robótico modelo *cyton I* com seu respectivo módulo de controle, e uma antena Wi-Fi. O suporte de alumínio foi customizado para que pudesse apoiar todos esses componentes eletrônicos e mecânicos associados ao computador embarcado.

3.2.1 Braço Robótico *Cyton* I

O braço robótico *cyton I* possui sete graus de liberdade, além de um *gripper* para manipulação de objetos. Cada eixo de rotação do robô é uma junta de revolução como na figura 2.11 que possui limites de rotação, e torque como apresentados na tabela 3.1. Todas as juntas possuem servos da fabricante *HiTec*. A identificação de cada junta pode ser visualizada na figura 3.4.

Junta	Limite de rotação	Torque máximo	Velocidade máxima	Modelo de servo
Shoulder Base	180 graus	2,42 N.m	7,48 rad/s	HS-805BB
Shoulder Pitch	170 graus	5,88 N.m	6,16 rad/s	2xHSR-5990TG
Shoulder Yaw	180 graus	2,42 N.m	7,48 rad/s	HS-805BB
Elbow Pitch	170 graus	2,94 N.m	6,16 rad/s	HSR-5990TG
Wrist Roll	180 graus	0,54 N.m	5,82 rad/s	HS-475HB
Wrist Yaw	130 graus	0,54 N.m	5,82 rad/s	HS-475HB
Wrist Pitch	135 graus	0,54 N.m	5,82 rad/s	HS-475HB

Tabela 3.1: Descrição das características básicas do braço robótico *Cyton* I

Usando a notação e os conceitos apresentados na seção 2.6, pode-se concluir que o espaço de configurações do braço é dado por

$$\mathcal{C}_{arm} = \mathbb{R}^1 \times \mathbb{R}^1 \times \mathbb{R}^1 \times \mathbb{R}^1 \times \mathbb{R}^1 \times \mathbb{R}^1 \times \mathbb{R}^1 = \mathbb{R}^7 \tag{3.1}$$

Isso porque todas as juntas de revolução possuem limites de rotação.

3.2.2 Kinect

O sensor kinect possui uma câmera RGB, um sensor de profundidade, um motor para movimentação do kinect, e um *array* de microfones. O sensor de profundidade consiste numa projeção de raios *laser* infravermelhos no ambiente.

Nesse trabalho, a grande contribuição do kinect é a produção de uma nuvem de pontos criada a partir do sensor de profundidade e que é usada como entrada para criação de um *OctoMap*.

3.3 Simulação

No processo de utilização de um robô é muito importante que exista uma etapa para simular computacionalmente o robô junto a tantos módulos de software quanto possível antes de testar o robô em um ambiente real.

Ao usar uma etapa de simulação diminui-se consideravelmente o número de erros nas primeiras tentativas de uso do robô. Além disso, muitos robôs possuem potencial para levar riscos de destruição ao ambiente, e inclusive a humanos, se não forem controlados de forma adequada, de modo que em muitas aplicações não existe muito espaço para erro. Nesses casos, a simulação do robô para ambiente em que ele será usado é crucial para agilizar a validação de um novo processo.

Nesse trabalho, foram usados alguns *softwares* e *frameworks* para produzir a simulação do robô. Nas próximas seções, será feita uma descrição sobre a importância e funcionalidade de cada um dos *softwares* ao mesmo tempo que é introduzido o processo de desenvolvimento da plataforma de simulação.

3.4 ROS

ROS é uma sigla para "Sistema Operacional para Robôs". Apesar do nome, ele não é um sistema operacional tradicional com a missão de escalonar e administrar tarefas, mas sim um *middleware*, uma camada extra sobre o sistema operacional do computador que serve para gerenciar e facilitar a comunicação entre vários arquivos executáveis sendo executados na máquina do servidor ou em outros computadores clientes.

ROS é um *framework* utilizado para desenvolver *software* para robôs. Nesse trabalho, todas as simulações e código para os robôs foram implementados visando o seu uso com ROS.

Note que ROS pode ser usado com robôs reais ou simulados.

Durante o desenvolvimento do ROS a principal preocupação dos criadores foi em facilitar a integração de *software* para robôs em larga escala[7]. Para que isso pudesse ser feito chegou-se a alguns requisitos de projeto que são descritos nas próximas seções.

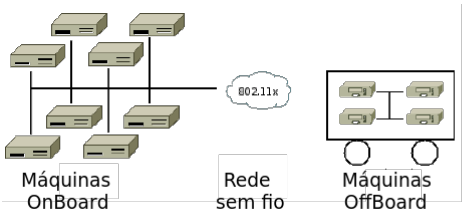


Figura 3.5: Exemplo de uma rede de computadores conectados através do ROS.[7]

3.4.1 P2P

Uma das principais filosofias do ROS é a de facilitar a comunicação entre processos existentes entre diferentes máquinas, para isso é utilizada uma topologia de rede *peer-to-peer*, ou P2P (Fig. 3.5). Dessa maneira, o usuário pode ter um computador embarcado em um robô, sem a necessidade que o computador embarcado tenha um alto poder computacional, pois o robô poderia enviar através da rede somente as informações necessárias para o processamento de tarefas computacionalmente caras, como processamento de imagens, para máquinas dedicadas.

Essa topologia P2P é necessária pois se fosse usada uma topologia com um servidor central seria gerado um custo de comunicação muito alto quando conectadas várias máquinas à mesma rede.

3.4.2 Multilinguagem

A estrutura do ROS permite que o usuário utilize vários tipos de linguagem para produzir processos. Uma vez que o arquivo executável tenha sido construído, ROS não irá distingui-lo pelo tipo de linguagem em que foi desenvolvido.

Atualmente ROS suporta nativamente as linguagens de programação Python e C++, além de possuir suporte para LISP e Octave através das duas primeiras.

3.4.3 Leveza

Se queremos que o ROS seja um contexto comum a todas as maquinas na rede, inclusive computadores embarcados com baixo poder computacional, é necessário que ROS seja extremamente leve.

Todas as tarefas computacionalmente caras são realizadas dentro de bibliotecas que estão integradas ao ROS mas que não fazem parte do núcleo do ROS. Isso também possibilita uma facilidade para desenvolver códigos e algoritmos escaláveis para outros computadores/robôs devido ao fato que a maior parte do problema de portar código para outros robôs está em usar esse código em outro sistema operacional, outro *firmware*, outro tipo de contexto. A partir do momento que o ROS vira um contexto de desenvolvimento comum, esse problema é minimizado grandemente.

3.4.4 Conceitos Fundamentais de ROS

Existem cinco conceitos fundamentais para o entendimento e uso do ROS como *framework* de trabalho:

- Nós;
- Mensagens;
- Tópicos;
- Serviços;
- Ações.

É comum representar as comunicações entre processos executáveis através de grafos, de forma que cada processo é então um nó de um desses grafos.

Os nós podem se comunicar através de mensagens. Uma mensagem é uma estrutura de dados definida no ROS e que possui uma interface para as linguagens suportadas pelo ROS.

Um nó recebe e manda mensagens através de tópicos. Um tópico funciona como uma caixa de correio. Vários processos podem "deixar" uma mensagem no tópico e vários outros processo podem "pegar" uma mensagem do tópico. No contexto de ROS isso significa que vários processos publicam uma mensagem em um tópico e que vários processos se inscrevem em um tópico. Apesar da simples analogia com uma caixa de correio funcionar bem para entender o conceito de tópico, ela não é perfeita pois tópicos em ROS podem receber somente um tipo de mensagem previamente especificado, enquanto caixas de correio recebem de documentos importantes a vários tipos de mala direta todos os dias.

A figura 3.6 mostra um exemplo em que o nó "/joint_state_publisher" publica uma mensagem no tópico "/joint_states" e o nó "/robot_state_publisher" se inscreve para receber a mensagem no tópico "/joint_states".

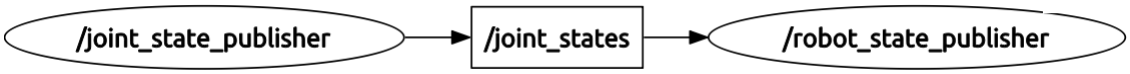


Figura 3.6: Exemplo de comunicação entre nós através de tópicos

A comunicação através de tópicos apesar de ser vantajosa para alguns casos, como em uma comunicação com vários nós ao mesmo tempo, pode demorar mais tempo para alguns tipos de aplicação. Para esse último caso existe uma estrutura de servidor/cliente dentro do ROS onde um nó pode agir como servidor, esperando uma requisição de um nó-cliente para agir. Quando isso acontece, diz-se que o nó-cliente requisitou um serviço do nó-servidor. Essa comunicação entre nós é feita de forma direta.

Existem também casos em que os serviços realizados demoram uma quantidade de tempo relativamente grande para uma aplicação. No entanto, enquanto o serviço é executado é possível que aconteça algum imprevisto, ou que o cliente necessite de um *feedback* do nó-servidor com

respeito ao serviço sendo executado. Para isso foram criadas as ações, que são basicamente serviços que podem ser cancelados pelo nó-cliente, e que também podem retornar valores de *feedback* para o nó cliente enquanto o serviço é executado.

Um bom exemplo para a necessidade de ações é um serviço que controla a execução de uma trajetória por um atuador. Se o controlador não está funcionando corretamente (e o nó-cliente só pode saber disso se houver *feedback* do nó-servidor) o nó-cliente tem que ter a opção de cancelar o serviço, pois um serviço sendo executado incorretamente pode significar inclusive perigo imediato para as pessoas ao redor do atuador!

3.4.5 Pacotes em ROS

Todo o software usando no *framework* ROS é organizado através de pacotes. Pacotes em ROS são basicamente pastas que organizam arquivos e documentação para um determinado tipo de aplicação. A estrutura mais comum de um pacote pode conter as seguintes pastas e arquivos:

- include/nome_do_pacote/ : Contém arquivos de cabeçalho;
- msg/ : Contém definições de mensagens criadas nesse pacote;
- src/ : Contém o código fonte de nós do pacote;
- srv/ : Contém definição de serviços/ações criados nesse pacote;
- launch/ : Contém arquivos *launch,* que quando chamados podem configurar parâmetros e executar nós;
- scripts/ : Scripts executáveis;
- packages.xml : Arquivo de definição do pacote;
- CMakeLists.txt : Arquivo que define a compilação do pacote como um todo (mensagens, serviços, código-fonte, etc.).

Existem vários tipos de pacotes disponibilizados como código aberto na comunidade de ROS.

3.4.6 Pacote TF

Dentre todos os pacotes de ROS, um dos mais importante é o pacote TF (*Transformation Frame*). Ele é usado para rastrear eixos de coordenadas e posições de corpos rígidos.

O TF funciona mantendo árvores de eixos de coordenadas. Um eixo de coordenadas tem sempre uma matriz homogênea de transformação para outros eixos na mesma árvore. De modo que mesmo que dois eixos de coordenadas não estejam conectados diretamente TF poderá calcular uma matriz de transformação homogênea, do tipo da equação 2.23, entre eles desde que existam eixos de coordenadas intermediários, criando um método simples para que o usuário tenha as coordenadas de um eixo de coordenadas para o outro.

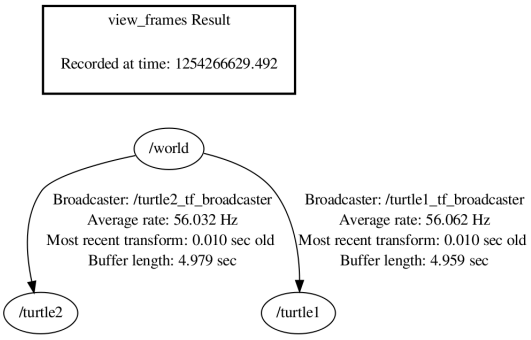


Figura 3.7: Visualização de uma árvore de eixos de coordenadas.

Um exemplo de árvore de eixos de coordenadas é dado na figura 3.7.

Como padrão a interface do pacote com o usuário é através de um tópico chamado "/tf" que mantém todos os eixos de coordenadas criados.

3.5 Simulação física

A princípio por esse se tratar de um trabalho em que o foco está voltado para algoritmos de planejamento de rotas, um software com visualização do *porthos* que aceitasse ações de controle tal qual o *porthos* seria suficiente para mostrar os algoritmos de planejamento. No entanto, com essa abordagem o trabalho desenvolvido não seria facilmente escalável para uma situação real. Para que a simulação seja pertinente para validação de algoritmos de planejamento que possam ser usados na vida real, faz-se necessário que a simulação leve em conta aspectos dinâmicos de simulação. Com essa finalidade o Gazebo foi escolhido para ser o simulador usado no trabalho.

O Gazebo está muito longe de ser um mero visualizador para um robô, mas ele possui recursos para reproduzir vários tipos de ambientes com gráficos 3D avançados suportados por um *engine open-source* chamada OGRE (Object-oriented Graphics Rendering Engine). Ele é especialmente poderoso para simulação dinâmica de objetos de tal forma que todos os objetos devem ter características físicas como massa, momento de inércia, velocidade, ou atrito para serem simulados, isso para que o simulador possa produzir uma simulação que permita que os objetos possam interagir entre si o mais realisticamente possível.

Pelo Gazebo é possível aplicar forças em qualquer ponto de um robô ou objeto, ou torques diretamente nas suas juntas. Praticamente qualquer aspecto da simulação como gravidade, condições de luz, ou coeficientes de atrito podem ser ajustados previamente, ou durante a simulação[13].

Todo esse cuidado em simular corretamente o ambiente faz com que sensores virtuais que estejam conectados a simulação possam receber valores muito parecidos com o que se espera na realidade. Para facilitar ainda mais, sensores podem ser ajustados para ter medidas menos perfeitas adicionando-se ruído nas mesmas.

Não obstante, já existem pacotes em ROS que fazem uma interface para o Gazebo a partir do ROS, o que facilita o trabalho imensamente.

Com esses recursos utilizar o Gazebo para testar algoritmos de planejamento por meio da execução de tarefas por um robô simulado se torna um passo extremamente importante no caminho de usar os algoritmos em desenvolvimento numa plataforma robótica real.

3.5.1 Arquitetura do Gazebo

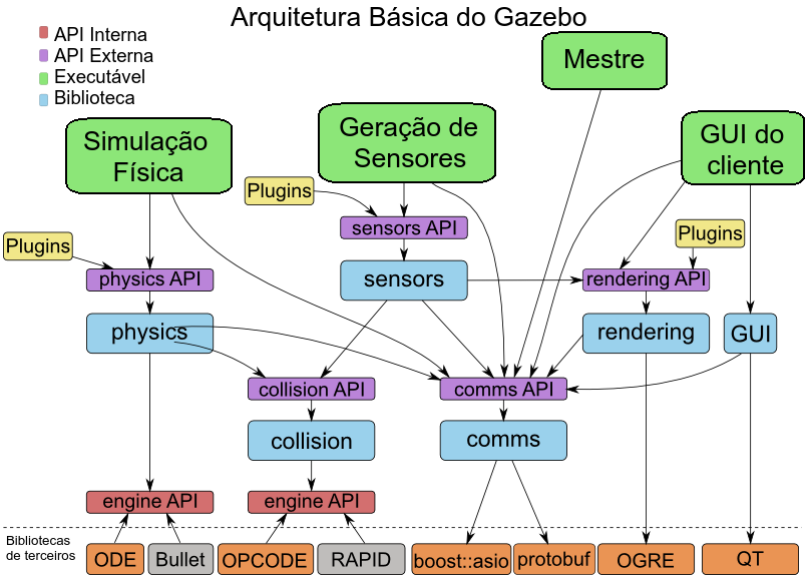


Figura 3.8: Arquitetura básica do Gazebo com seus quatro processos principais: Simulação Física; Geração de sensores; Interface gráfica para o usuário; e um processo mestre para coordenação entre os outros três.

A arquitetura do Gazebo é distribuída em três processos principais, interface gráfica, simulação física e geração de sensores, que são coordenados por um processo mestre.

No nível mais baixo da arquitetura se encontram bibliotecas de terceiros para que modelos criados para uso no Gazebo não fiquem reféns de mudanças em ferramentas específicas para simulação, renderização, etc. Isso também é uma forma de garantir que um usuário mais avançado possa usar outras ferramentas que não as instaladas originalmente, sem quebrar paradigmas da arquitetura original.

3.5.2 Descrevendo um robô

O modo padrão de se descrever um robô em ROS é através de um arquivo em formato XML do tipo URDF, ou *Unified Robot Description Format*.

URDF é capaz de descrever robôs formados por corpos rígidos conectados por juntas (Essas partes são as mesmas $\mathcal{A}_i$ citadas na seção 2.6.4). Atualmente URDF é capaz de descrever um robô pelas seguintes características:

- Localização dos componentes;
- Componentes geométricos para colisão;

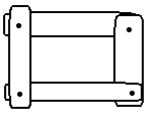


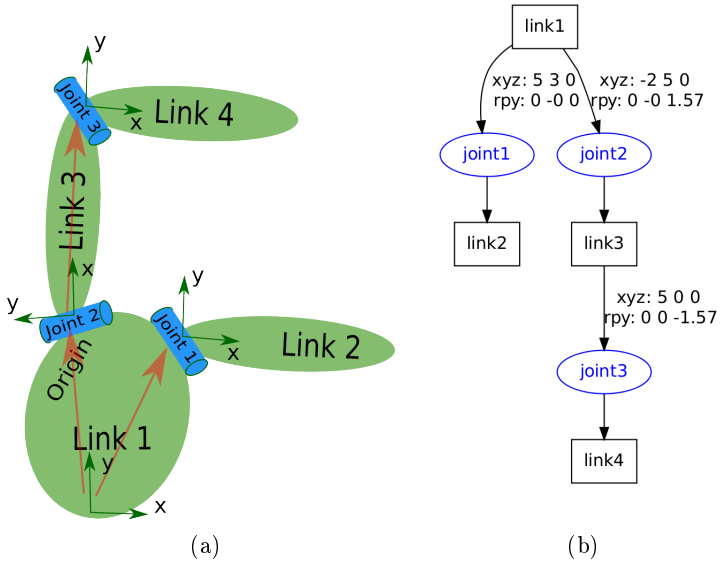
Figura 3.9: Quatro barras conectadas entre sí

- Componentes geométricos para visualização;
- Tipos de Junta;
- Propriedades Físicas;
- Materiais.

Também é importante mencionar o que URDF *não* consegue descrever:

- Quatro barras conectadas entre si (figura 3.9);
- Engrenagens;
- Polias/Cordas;
- Materiais flexíveis

Um robô é descrito como uma árvore que possui cadeias de *links* conectados por juntas, de modo que a origem de cada *link* se encontra na junta em que o *link* pai (mais acima da árvore) conecta-se ao *link* filho.



(a) (b)

Figura 3.10: Como as coordenadas de um "robô" são descritas usando URDF

3.5.3 Integração ROS e Gazebo

Antes de descrever a integração entre ROS e Gazebo faz-se necessário um parênteses sobre a descrição de um robô no Gazebo. Para simular um robô de forma satisfatória o Gazebo necessita de uma descrição mais detalhada do robô do que a descrição dada no modelo URDF, por isso o Gazebo usa outro tipo de arquivo de descrição de robôs, SDF, ou *Simulation Description File*. Dentre vários aspectos que o SDF estende o uso de URDF, o mais notável deles é a presença de uma descrição de sensores compatíveis com o ambiente de simulação do Gazebo.

SDF é outro tipo de descrição de um robô, sendo que se o robô já possuir uma descrição por SDF, não é necessário que ele possua uma descrição em URDF para que se faça a simulação no Gazebo.

A integração entre ROS e Gazebo é feita através de um conjunto de pacotes, ou meta-pacote, chamado de "gazebo_ros_pkgs" (fig.3.11).

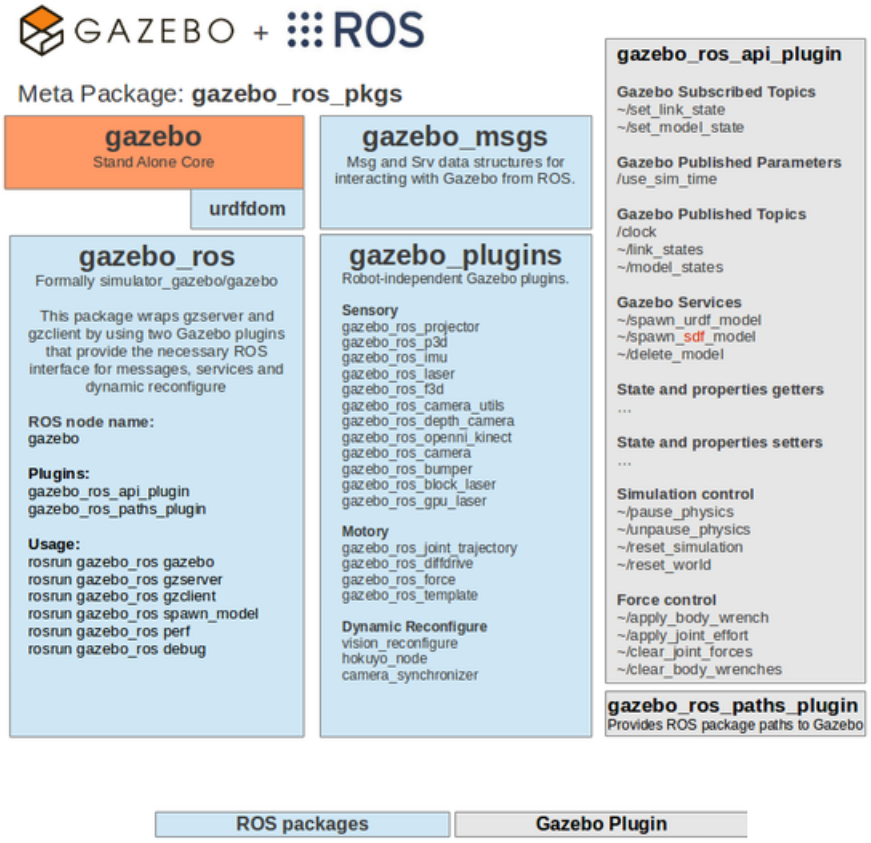


Figura 3.11: Meta-pacote "gazebo_ros_pkgs" para integração entre Gazebo e ROS

Esse meta-pacote permite "colocar" um robô no mundo do Gazebo por meio de uma descrição URDF. O serviço que realiza essa tarefa na realidade traduz o URDF do robô em questão para o formato nativo do Gazebo, o SDF, sendo que os recursos a mais do SDF são especificados em campos especiais dentro do URDF, cuja única função é permitir compatibilidade do URDF com o Gazebo.

Além de "traduzir" robôs do ROS para o Gazebo, esse meta-pacote também publica mensagens, e disponibiliza serviços para interação com o robô simulado no gazebo diretamente com o ROS.

3.5.4 Pacote ROS Control

É possível controlar atuadores através das estruturas mais básicas do ROS, como tópicos e serviços. No entanto ao fazê-lo dessa maneira não há garantias quanto a execução em tempo real, ou tempo de simulação, das ações de controle, ou mesmo do recebimento das medidas dos sensores.

O meta-pacote ROS Control implementa um *framework* dentro do ROS para uso de controladores, garantindo que todos os seus requisitos de tempo sejam satisfeitos .

ROS Control recebe como entrada dados dos encoders dos atuadores dos robô para saber as posições atuais das juntas, além disso também recebem uma entrada para uma ação de controle. Ele usa um mecanismo genérico de alimentação onde geralmente são implementados controladores PID para controlar uma saída que tipicamente é um torque aplicado a uma junta por um dos atuadores.

ROS Control é um pacote relativamente novo, ainda em desenvolvimento, mas os seus atuais recursos já permitem controlar o manipulador do *porthos* por exemplo.

O pacote ROS Control também possui uma interface com os pacotes ROS do Gazebo, de modo que fica fácil implementar controladores para um robô simulado dentro do Gazebo (Fig. 3.12).

3.5.5 Descrevendo o *Porthos*

O primeiro passo para desenvolver uma simulação satisfatória para o robô é descrever o robô dentro de um arquivo URDF.

URDF permite que usuário descreva partes do seu robô como formas geométricas simples como cilindros, ou caixas por exemplo. Além disso é possível especificar arquivos do tipo STL que podem modelar formas geométricas tão complexas quanto possível. Isso é exatamente o que precisamos aqui, pois se fossem usadas descrições muito fora da realidade do robô o planejamento poderia falhar de diversas maneiras.

Cada parte do robô é modelada então por um arquivo STL. Para construir esse modelo foi usado em *plugin* para o *software SolidWorks* chamado *SolidWorks to URDF Exporter*. Esse *plugin* permite que o usuário exporte um desenho 3D criado no *SolidWorks* para o formato URDF.

Durante o processo de exportação o *plugin* em questão cria um pacote ROS que contém não só um arquivo URDF, como também os arquivos STL para visualização do robô e checagem de colisões com base em uma montagem de um arquivo ASM (nativo do *SolidWorks*).

Durante o processo de desenho é necessário criar partes menores que não necessitam ser explicitadas no arquivo URDF, por isso, o arquivo ASM principal é composto de vários arquivos ASM secundários, de forma que o *plugin* converta toda uma montagem secundária como uma parte do robô (Fig. 3.13).

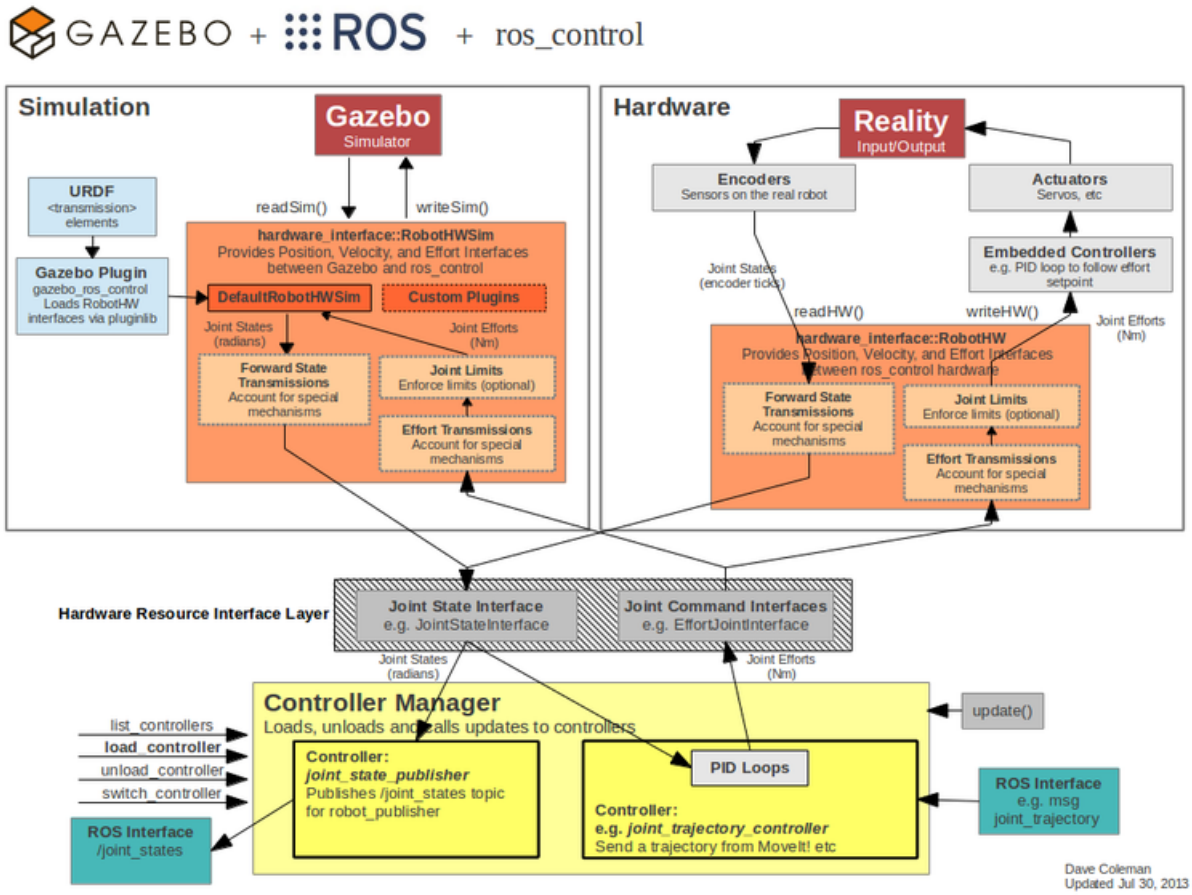


Figura 3.12: Diagrama da arquitetura do pacote ROS Control

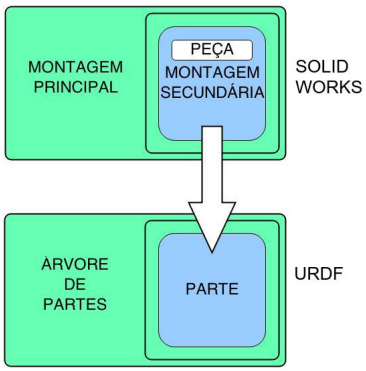


Figura 3.13: Conversão de arquivos em *SolidWorks* para um pacote ROS com descrição URDF

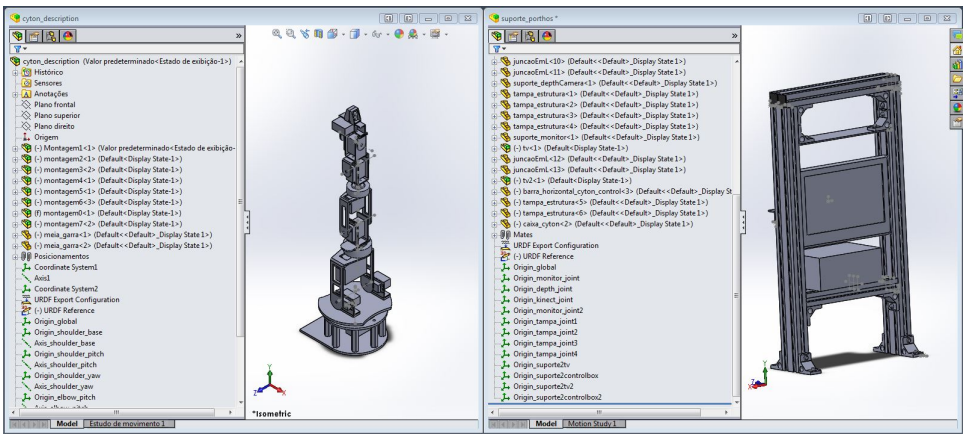


Figura 3.14: Modelos 3D do *Cyton I*, e do suporte de alumínio customizado para os periféricos do *Porthos* no *SolidWorks*.

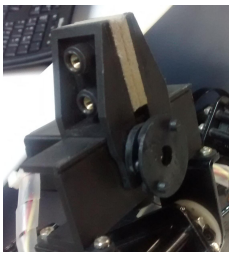


Figura 3.15: *Gripper* do *Cyton I*

Além de converter peças e montagens do *SolidWorks* para um formato STL, o *plugin* também é capaz de encontrar automaticamente eixos de rotação para cada montagem.

Os robôs da família *Pioneer* já possuíam modelos (inclusive URDF e SDF) disponíveis na comunidade de ROS, portanto para esse trabalho só foi necessário desenhar os modelos 3D do manipulador *Cyton I* e dos periféricos ligados ao suporte do Porthos, com exceção do *kinect* que assim como o *Pioneer* já possuía modelos disponíveis para uso (Fig. 3.14).

3.5.5.1 Modelagem do *Gripper*

O *gripper* do manipulador abre e fecha conforme o movimento do servo associado a ele.

As "garras" , esquerda e direita, se movem como se estivessem em uma junta linear, mas a atuação é realizada sobre uma junta de revolução, isso só é possível por causa da estrutura na figura 3.15.

No entanto, esse tipo de relação com engrenagens, como foi citado na seção 3.5.2, não é suportado por um modelo URDF. Para fins de simulação aqui vamos modelar o *gripper* como sendo composto de dois *links,* uma "garra" esquerda e outra direita sendo que ambas são atuadas independentemente em juntas lineares.

3.5.5.2 Visualização e verificação do modelo

Feito os desenhos e a modelagem do Porthos o próximo passo é de verificar se eixos de atuação de cada junta estão respondendo corretamente, isto porque um eixo de atuação de uma junta de revolução que esteja na direção oposta a direção correta pode significar que a junta gire no sentido contrário ao desejado.

A melhor maneira de verificar isso é usando o pacote padrão de visualização do ROS, o RViz.

A figura 3.16 mostra uma aplicação com uma visualização do Porthos dentro do RViz sendo comandada por um nó que publica as posições das juntas.

Para entender melhor a aplicação usamos uma ferramenta nativa do ROS, o *rqt_ graph,* que faz uma visualização do que está acontecendo no ambiente do ROS em um instante de tempo. Essa ferramenta também foi utilizada para criar a figura 3.6, e será usada frequentemente no restante desse trabalho.

3.5.6 Simulando o Porthos

Para usar o Porthos simulado dentro do gazebo é necessário usar as interfaces do gazebo com o ROS. Para a base móvel será usado diretamente um plugin para controle da direção diferencial do robô, e para o manipulador será usada a interface com os controladores do pacote ROS Control. Ambas são descritas nas seções a seguir.

3.5.6.1 Controle da base móvel

A base móvel do Porthos é uma base com direção diferencial. As únicas variáveis necessárias para construir o modelo derivado na seção 2.8 são a distância entre as rodas, $l = 40cm$, e o diâmetro das rodas, $2r = 21,5cm$. O Gazebo recebe essas informações através um campo especial dentro do URDF do *Porthos.*

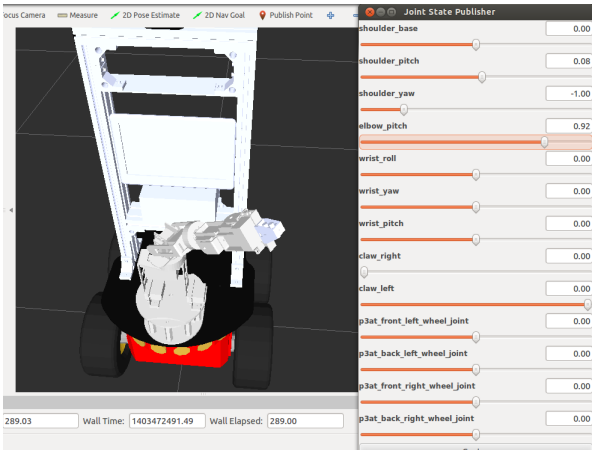
Quando a aplicação de simulação é iniciada através do ROS, o nó "Gazebo" passa a se inscrever para receber mensagens do tópico "/cmd_vel" que encapsula comandos de velocidade linear , v, e de velocidade angular, $\dot{\theta}$.

Ao passo que recebe comandos de velocidade para mover o robô no ambiente de simulação, o nó "gazebo" também publica as novas posições do robô ao longo do tempo no tópico "/tf" para que demais aplicações no ROS possam saber das novas configurações da base móvel.

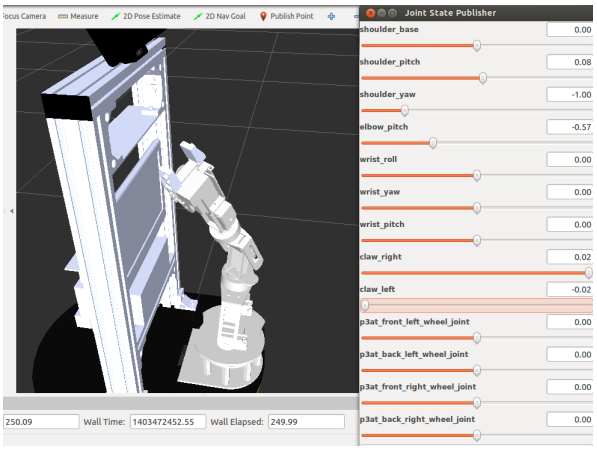
Para ilustrar o controle da base móvel foi feita uma aplicação aonde um *joystick* envia comandos de velocidade, através de um nó "joy_node", para o robô simulado (fig. 3.18).

3.5.6.2 Controle do manipulador

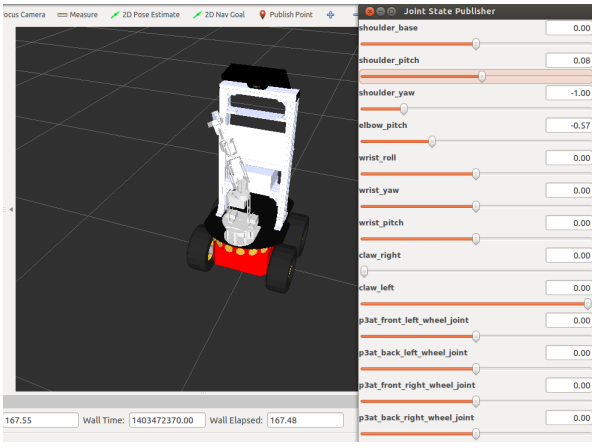
Usando o pacote ROS control pode-se implementar controladores do tipo PID para uma junta em robô simulado no Gazebo. Aqui, cada junta do manipulador (o braço robótico *cyton I*) possui um controlador PID associado. Além disso, as juntas virtuais do *gripper* também possuem controle



(a)



(b)



(c)

Figura 3.16: Aplicação para visualização e verificação do modelo URDF do *Porthos*

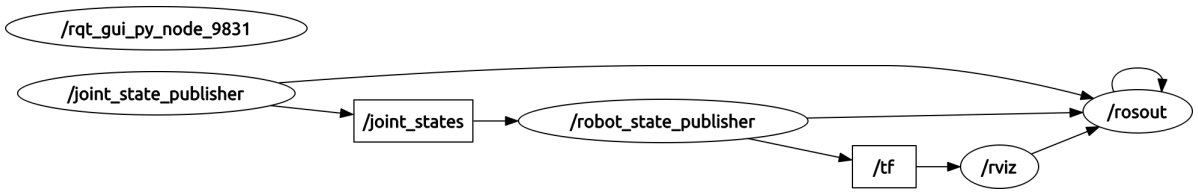


Figura 3.17: Grafo da aplicação para verificação do modelo URDF

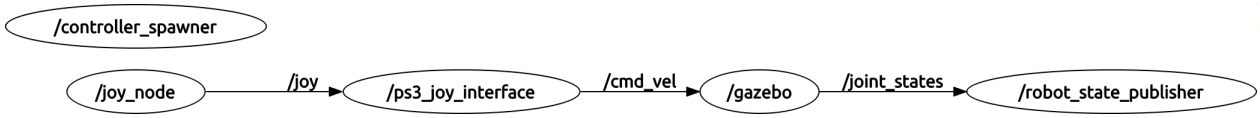


Figura 3.18: Grafo da aplicação para controle da base móvel através de um *joystick*

PID com anti-*Windup* do canal integral, principalmente para diminuir o sobrepasso nas atuações das juntas. Os controladores foram ajustados empiricamente de modo que produzissem uma melhor resposta em termos de maior velocidade, menor erro, e menor sobrepasso.

Os ganhos PID, assim como o limitador de ganho integral (I_{clamp}), dos controladores são mostrados na tabela 3.2.

Nome da Junta	P	I	D	I_{clamp}
Shoulder_base	1	0,6	0,3	-
Shoulder_pitch	4	2,5	1,8	10
Shoulder_yaw	4	3	0,05	10
Elbow_pitch	3	2	0,5	10
Wrist_roll	4	2	0	10
Wrist_yaw	4	1	0	10
Wrist_Pitch	3	1	0	10
Claw_left	4	3	2	1
Claw_right	4	3	2	1

Tabela 3.2: Ganhos PID para cada junta do manipulador simulado.

3.6 Planejamento e execução de trajetórias

3.6.1 OMPL

Ao pesquisar e desenvolver novos algoritmos de planejamento é necessário comparar a performance do algoritmo novo com outros algoritmos da literatura. E para isso é preciso implementar vários algoritmos num mesmo módulo de testes.

Foram explicados alguns dos principais conceitos sobre algoritmos para planejamento de rotas baseado em amostragem de um espaço de configuração, aonde muitos algoritmos, como o RRT por exemplo, possuem algoritmos simples e de relativamente fácil implementação, como foi feito em [12]. No entanto, vários algoritmos dessa classe são de difícil implementação, o que pode

levar uma tarefa aparentemente simples de comparação de algoritmos a se tornar um gargalo no desenvolvimento de novas tecnologias.

Nesse trabalho que tem como objetivo maior construir uma plataforma de simulação para o Porthos, ao invés de usarmos uma solução customizada de implementação de vários algoritmos de planejamento, decidiu-se por utilizar a biblioteca OMPL (*Open Motion Planning Library*) que implementa vários algoritmos de planejamento de rotas do atual estado da arte, por dois motivos principais, o primeiro é a facilidade de testar e ajustar parâmetros de algoritmos conhecidos; e o segundo é o de facilitar implementações de novos algoritmos por pesquisadores do LARA criando a plataforma de simulação com base numa biblioteca que é usada frequentemente pela comunidade de pesquisa em planejamento de rotas e trajetórias [15].

A biblioteca OMPL também possui no seu repositório de código aberto uma interface gráfica simples que possibilita uma fácil interface para pesquisadores, a OMPLapp. A figura 2.19, por exemplo, foi produzida usando essa interface gráfica.

Além disso, a biblioteca OMPL, assim como o Gazebo, é facilmente integrável com o ROS, possuindo inclusive um pacote para planejamento e execução de trajetórias de um robô, real ou simulado, que use ROS. O nome desse pacote é "MoveIt!".

3.6.2 MoveIt!

A arquitetura da biblioteca OMPL é organizada de forma que seja possível usá-la nas mais diversas aplicações para planejamento de rotas. Esse aspecto generalista da biblioteca faz com que não seja possível utilizá-la de forma direta para planejar rotas para robôs. Para que isso seja feito, é necessário prover módulos de colisão de objetos, módulos de cinemática inversa, etc.

O MoveIt! é uma coleção de pacotes para o ROS que encapsula *software* para planejamento de trajetórias, manipulação, cinemática inversa e direta, controle e navegação de robôs[14].

Assim como o Gazebo, o MoveIt! possui uma arquitetura que possibilita ao usuário prover vários módulos customizados, ou de terceiros, que possam ser mais úteis para uma aplicação específica do que os módulos originais (Fig. 3.19). É possível inclusive usar alguma outra biblioteca de planejamento de trajetórias além da OMPL se desejado.

O MoveIt! possui um módulo padrão de cinemática inversa que possibilita ao usuário planejar para uma posição do manipulador, assim como para todas as juntas.

Sendo o MoveIt! um pacote do ROS, é natural que ele use a descrição do robô dada pelo arquivo URDF do robô. No entanto, como já foi discutido anteriormente, o URDF possui algumas limitações para descrição da dinâmica do robô, e como no caso do Gazebo, o MoveIt também possui uma descrição para o robô a parte do URDF, com um arquivo SRDF, com a diferença básica de que o SRDF é um arquivo que complementa informações do arquivo URDF, com informações especialmente úteis para o planejamento de rotas e trajetórias. Esse arquivo SRDF precisa ser criado antes de qualquer tentativa de utilização do MoveIt!.

O arquivo SRDF inclui informações sobre para quais grupos de *links* do robô o MoveIt! poderá planejar, (*e.g.* se o robô possuir dois braços, o MoveIt! poderá produzir planos separados para

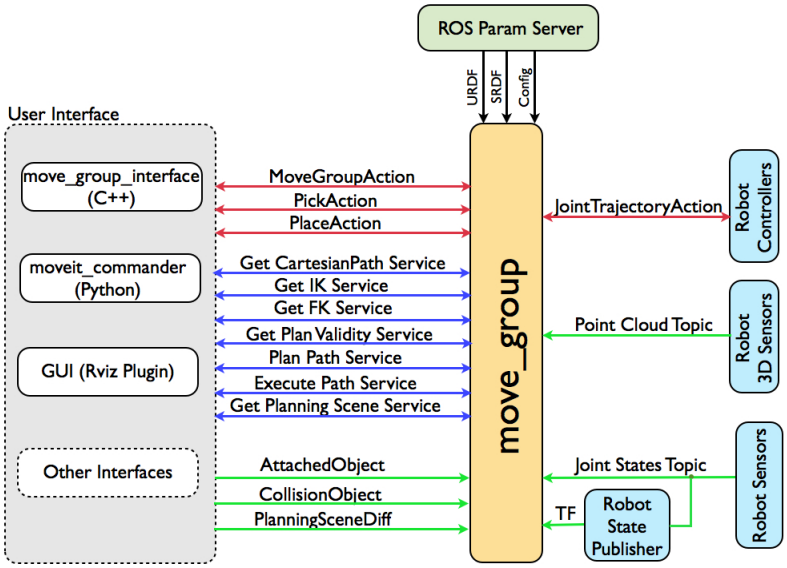


Figura 3.19: Arquitetura do pacote "MoveIt!"

cada grupo de *links*, braço 1 e braço 2), sobre quais *links* possuem um manipulador, e também possui uma representação de uma matriz de colisões permitidas, ou ACM em inglês.

A ACM é um dos conceitos mais importantes associados ao MoveIt!, pois ela decodifica em valores binários se dois corpos rígidos precisam ser checados se estão em colisão ou não. Se dois corpos estiverem muito longe entre si de tal forma que nunca entrarão em colisão, então a ACM revelará um valor 1 entre os dois por exemplo.

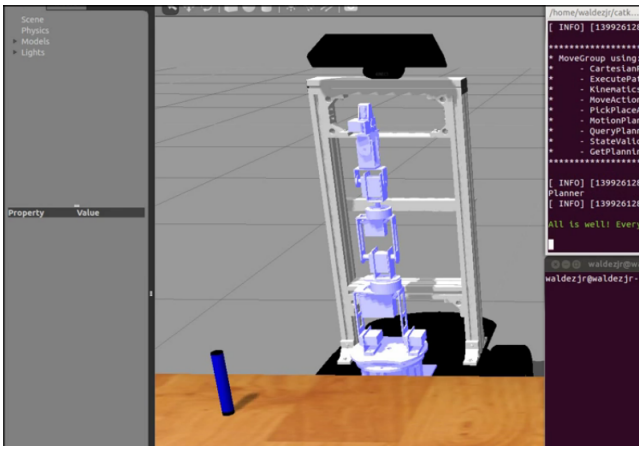
A ACM é uma maneira de melhorar a eficiência do módulo que verifica se uma determinada configuração está ou não dentro de $\mathcal{C}_{obs}$. A sua existência é justificada pelo fato de que a verificação de colisões dura aproximadamente 90% do tempo de planejar uma rota, ou trajetória.

O MoveIt! se inscreve no tópico "/tf" para ouvir as posições do robô, portanto é necessário que o robô tenha um nó que publique as suas posições. Além de publicar suas posições, outro requisito para usar o MoveIt! com um robô é que o robô possua um nó servidor de ações para controlar as suas juntas do tipo "JointTrajectoryAction" como implementados na seção 3.5.6.2.

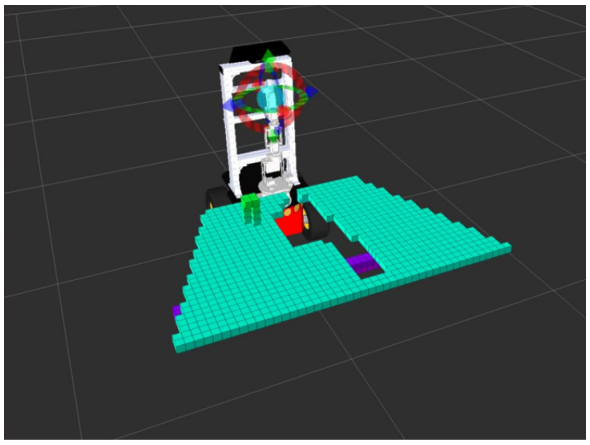
Outro recurso bastante interessante do MoveIt! é a possibilidade de mapear objetos através de *OctoMaps*. O MoveIt! ouve algum tópico que contenha mensagens que sejam nuvens de pontos, que são então usadas para construir *OctoMaps* exatamente como visto na seção 2.2.2(Fig. 3.20b).

Quanto a manipulação de objetos o MoveIt! possui um método aonde o usuário informa como o manipulador pode pegar um objeto. O MoveIt! então, planeja para que isso seja possível, e quando executa essa tarefa ele anexa o objeto a estrutura de planejamento do robô, de maneira que agora o MoveIt! fará futuros planejamentos de trajetória baseados no robô mais o objeto anexado (figuras 3.20c,3.20d).

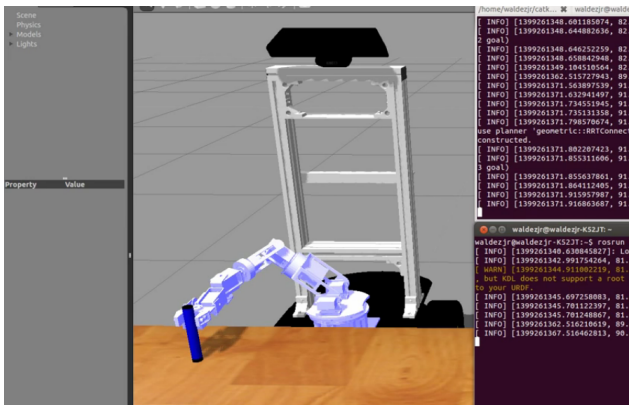
Apesar de poder tratar objetos, e de desviar de obstáculos no planejamento através do módulo de sensores juntamente com *OctoMaps*, o MoveIt! não consegue reconhecer objetos. O reconhecimento de objetos é uma tarefa que não se encontra dentro do *framework* do MoveIt! nem mesmo



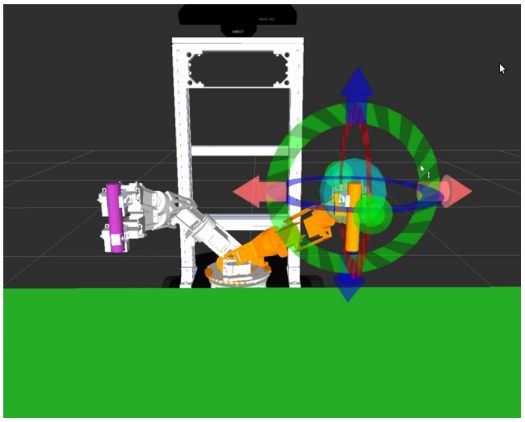
(a) *Porthos* simulado dentro do Gazebo.



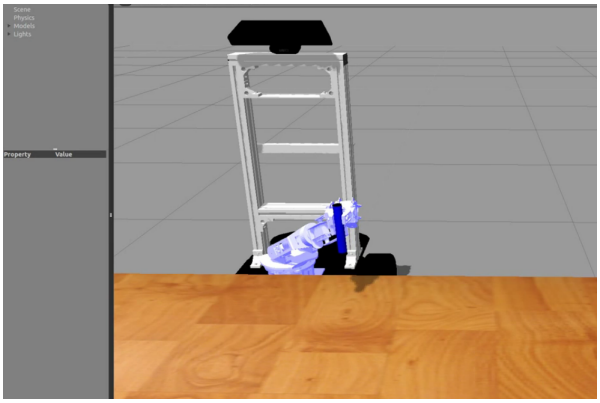
(b) Visualização do Porthos e de um *OctoMap* gerado pelo kinect simulado dentro do Gazebo.



(c) *Porthos* pegando uma peça no Gazebo.



(d) Planejamento de rotas com a peça anexada ao braço robótico.



(e) Resultado de planejamento do *Porthos* simulado.

Figura 3.20: Integração de *Porthos* simulado com o MoveIt!.

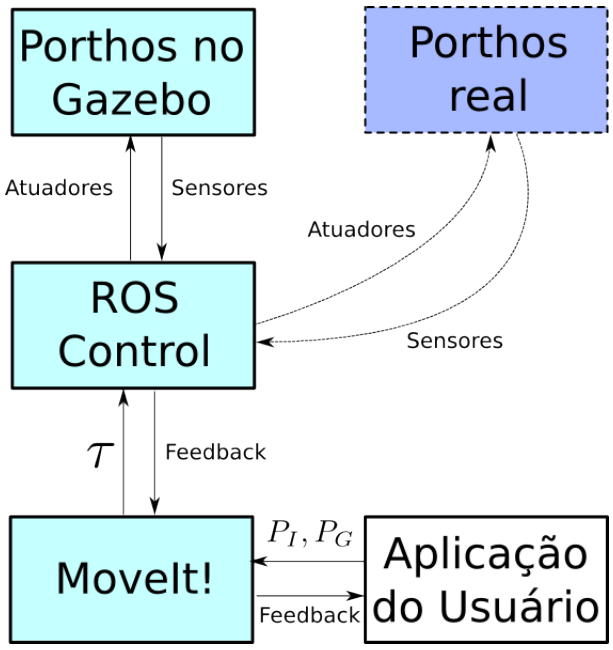


Figura 3.21: Arquitetura da aplicação desenvolvida para planejar rotas para o *cyton I*, o braço robótico montado no Porthos. Os sinais de *feedback* retornam informações sobre as requisições feitas tanto pelo usuário, quanto pelo MoveIt!

em módulos personalizáveis. É tarefa da aplicação do usuário reconhecer objetos e integrá-los ao planejamento no MoveIt!.

A maior limitação do MoveIt! no entanto, é que ele não possui uma interface genérica para utilização dos planejadores com restrições diferenciais (cinedinâmicas) da biblioteca OMPL. Isso faz com que não seja possível realizar um planejamento eficiente para uma base móvel com direção diferencial como a que o *Porthos* possui.

Para reverter esse problema foram implementadas classes para planejamento e execução de rotas calculadas pela OMPL. Essas classes são descritas na seção 3.6.4.

3.6.3 Plataforma desenvolvida para planejamento de rotas para o braço do Porthos

Nesse trabalho, foi desenvolvida uma solução que disponibiliza uma forma de testar e desenvolver algoritmos de planejamento de rotas (como os da seção 2.11) para o Porthos.

A figura 3.21 mostra a dinâmica entre esses pacotes e uma aplicação em ROS de algum usuário, onde pode-se perceber que a execução das rotas está desacoplada do planejamento realizado pelos planejadores do MoveIt!. Não obstante, é possível inferir que a solução não depende de se ter o Porthos simulado, ou o Porthos real. Isso só é possível devido aos esforços do Gazebo em oferecer uma arquitetura que simule um robô nos menores detalhes.

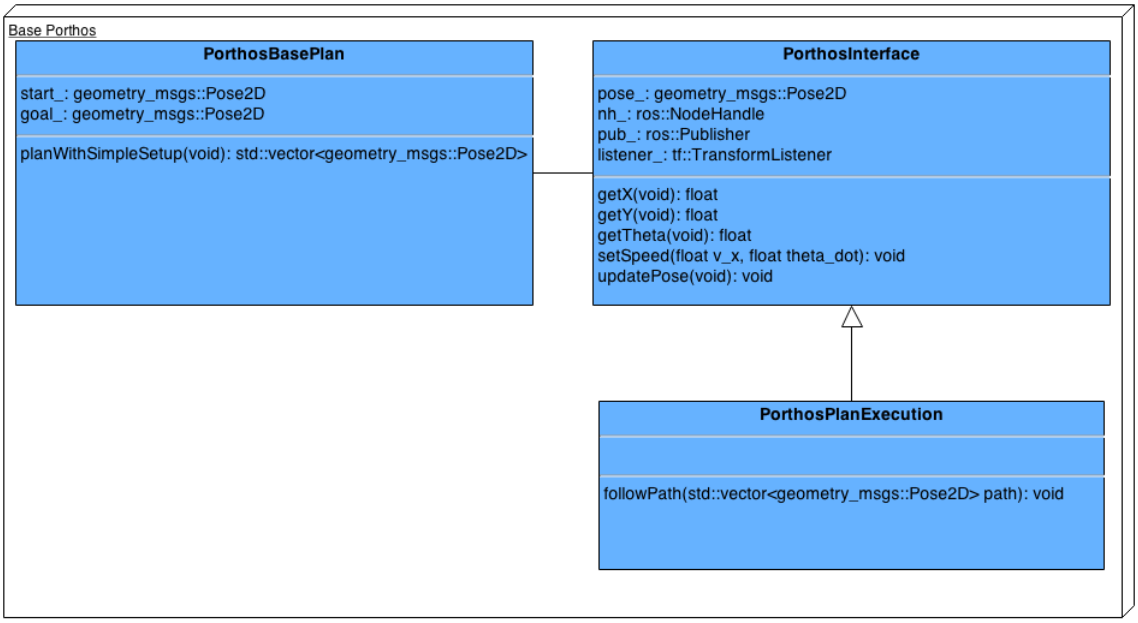


Figura 3.22: Classes para planejamento e execução de rotas da base móvel do *Porthos*.

3.6.4 Planejamento da base móvel

Foram criadas três classes para planejamento e execução de rotas pela base móvel.

Primeiramente, foi criada a classe "PorthosInterface" para que o *Porthos* simulado tenha uma interface genérica com o ambiente do ROS. Essa classe implementa métodos para ajustar velocidade linear e angular da base, assim como métodos para receber a atual configuração da base do robô em $SE(2)$. Essa classe possui inclusive, limitadores de velocidade, que impedem que o robô receba velocidades maiores do que o alcançável , uma certa velocidade V_{max}.

A classe "PorthosBasePlan" prima pela simplicidade, de maneira que encapsula dentro de uma função a configuração do espaço de configurações, do espaço de ações de controle, e o planejamento de uma rota que leve a configuração inicial do robô para uma configuração final desejada.

O módulo de detecção de colisões implementado é bastante simples, sendo realizado baseado nas técnicas da seção 2.2.1.

Na implementação, os espaços de ação de controle da velocidade linear e da velocidade angular foram limitados às regiões:

$$I_v = (0, V_{max}) \tag{3.2}$$

$$I_{\dot{\theta}} = (-\pi/4, \pi/4) \tag{3.3}$$

Ao limitar esses espaços de ação de controle as rotas calculadas para o robô se mostraram mais suaves, de maneira que não foi usada nenhuma técnica de pós-processamento para suavização das rotas, inclusive porque essas técnicas geralmente ignoram restrições cinemáticas do robô [12].

Apesar de ignoradas nesse módulo do projeto, as técnicas de suavização de rotas são muito

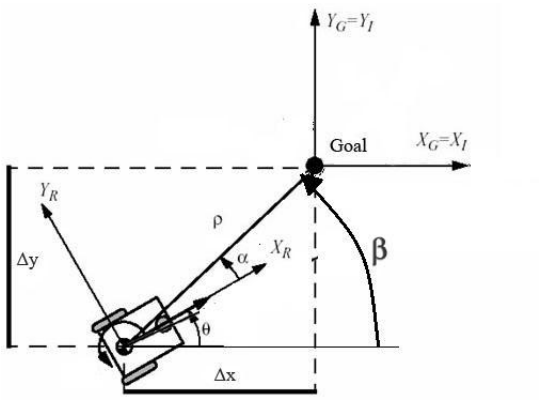


Figura 3.23: Robô seguindo uma trajetória que o leve até um objetivo em $\{X_G, Y_G\}$ com uma determinada orientação β.

importantes para otimizar o tamanho da rota no tempo, em casos que planejamento de rotas puramente geométrico é usado. Nesse trabalho, todas as aplicações desenvolvidas com o *MoveIt!* utilizam suavização de rotas por exemplo.

Na implementação do módulo de planejamento percebeu-se que eram encontradas soluções muito mais rapidamente usando o método de Euler para "avançar" os estados no tempo, do que quando era o usado o método de Runge-Kutta, o que já era esperado devido ao menor número de fórmulas para o método de Euler.

A classe "PorthosPlanExecution" herda as propriedades da classe "PorthosInterface" e implementa um método para receber e executar uma rota calculada em "PorthosBasePlan".

3.6.4.1 Execução de rota

A rota calculada na classe de planejamento é nada mais do que um *array* de posições e orientações que a base deve seguir. Para que a base móvel siga a rota é usada uma estratégia de execução:

1. Robô rotaciona até que $\alpha \approx 0$, com uma lei de controle $\dot{\theta} = K_\alpha.\alpha$;
2. Robô translada e rotaciona até que $\rho \approx 0$, com uma lei de controle $\dot{\theta} = K_\alpha.\alpha$ e $v = K_\rho.\rho$.

Aonde ρ, α podem ser observados na figura 3.23.

Essa estratégia é usada entre os pontos da trajetória até que o robô chegue a sua posição final. Para garantir que o robô esteja também com a orientação desejada é usada uma ação de controle final que corrige a orientação do robô, girando-o em torno do próprio eixo.

Para esse trabalho foram usados $K_\alpha = 5$ e $K_\rho = 1$.

Capítulo 4

Resultados experimentais

"A little less conversation, a little more action please"
Elvis Presley

Nessa seção, são apresentados resultados referentes a experimentos usados para verificar a eficácia dos módulos de execução e planejamento de rotas para a base móvel e para o braço robótico do *cyton I*.

4.1 Execução de trajetórias para o braço robótico

Para que se possa saber a performance dos controladores PID de cada junta, mostrados na seção 3.5.6.2, são mostradas as respostas de cada junta do braço robótico na forma de gráficos da posição desejada e atual da junta.

Para cada figura foi escolhida aleatoriamente uma configuração para o braço robótico, e então foi calculado um plano de trajetórias com o algoritmo RRT-Connect que posteriormente foi executado pelo controlador.

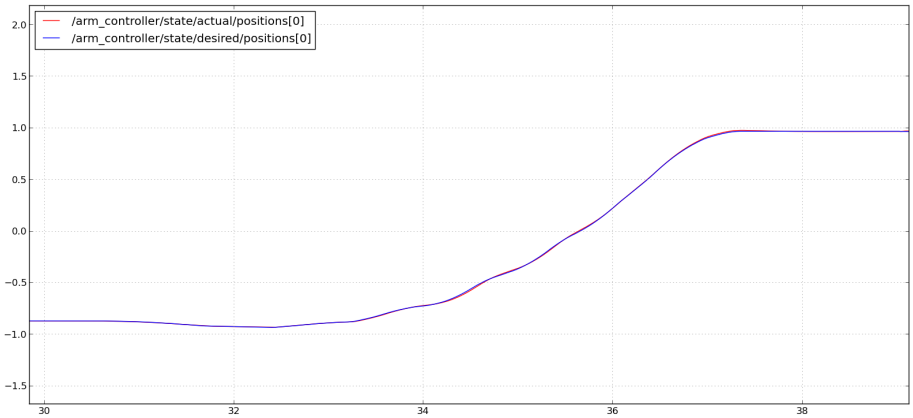


Figura 4.1: Posição da junta *Shoulder Base* (rad) X Tempo (s)

O gráfico da junta *Shoulder Pitch* (Fig. 4.2), apesar de várias tentativas de ajuste dos ganhos do controlador PID, foi o único com um sobrepasso significativo. Além disso, em algumas trajetórias

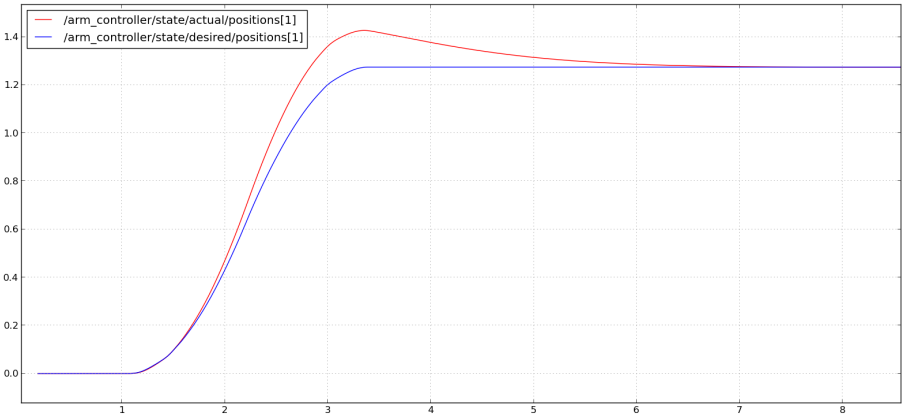


Figura 4.2: Posição da junta *Shoulder Pitch* (rad) X Tempo (s)

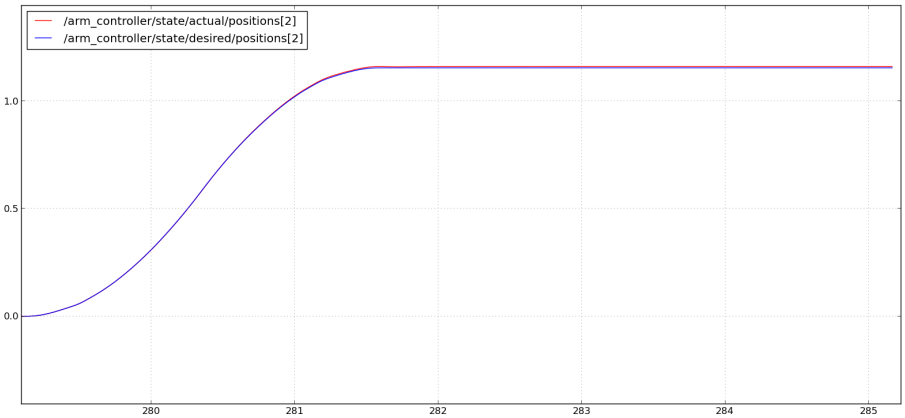


Figura 4.3: Posição da junta *Shoulder Yaw* (rad) X Tempo (s)

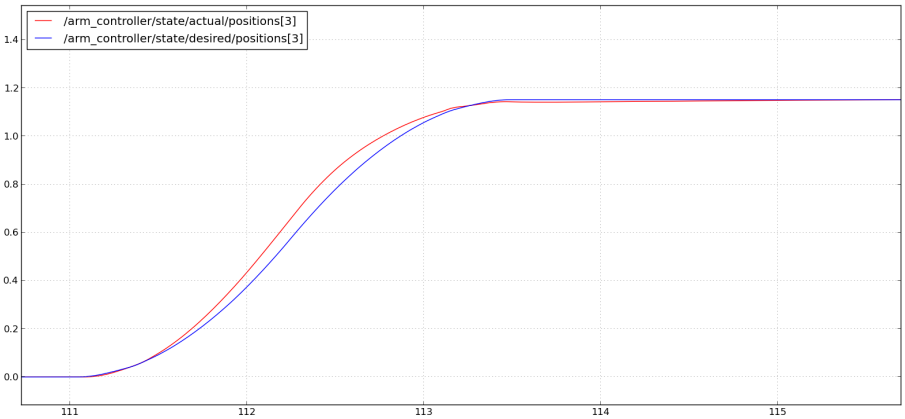


Figura 4.4: Posição da junta *Elbow Pitch* (rad) X Tempo (s)

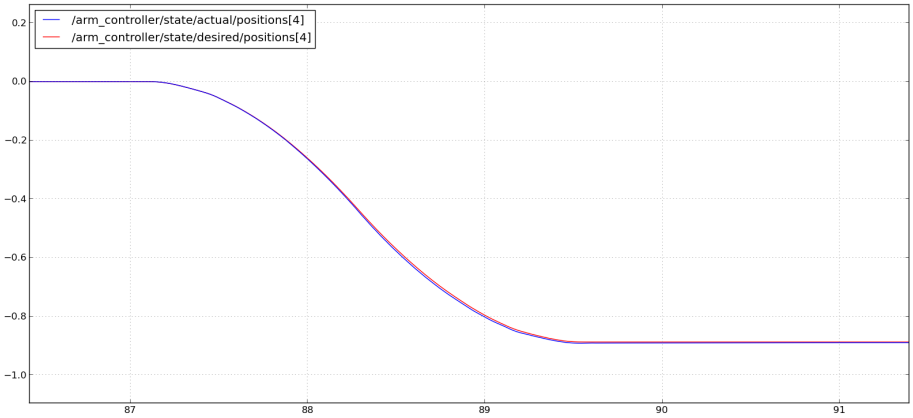


Figura 4.5: Posição da junta *Wrist Roll* (rad) X Tempo (s)

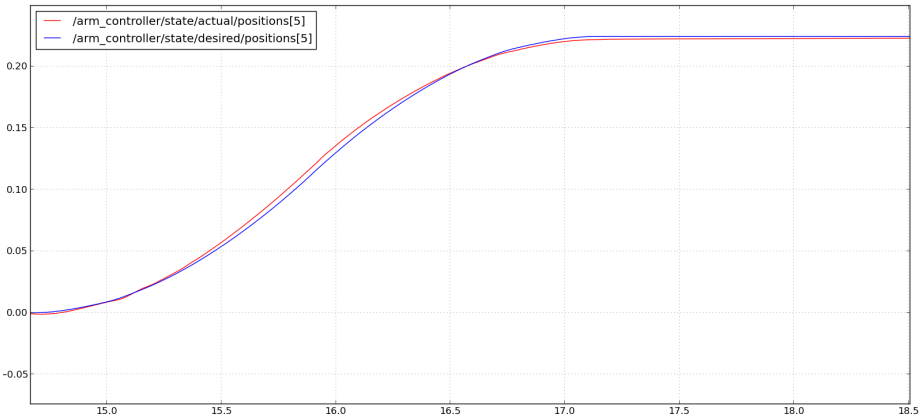


Figura 4.6: Posição da junta *Wrist Yaw* (rad) X Tempo (s)

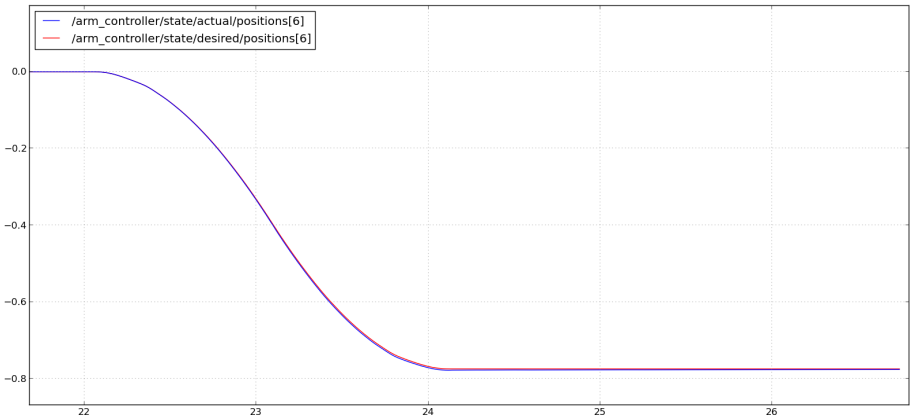


Figura 4.7: Posição da junta *Wrist Pitch* (rad) X Tempo (s)

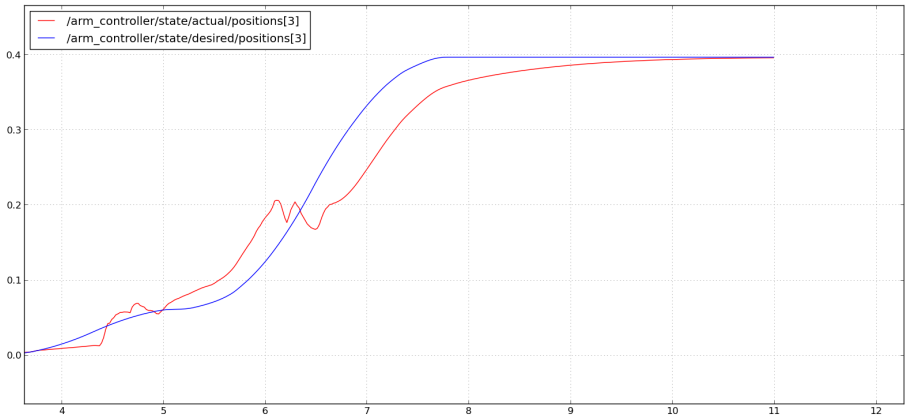


Figura 4.8: Posição da junta *Elbow Pitch* (com irregularidades) X Tempo (s)

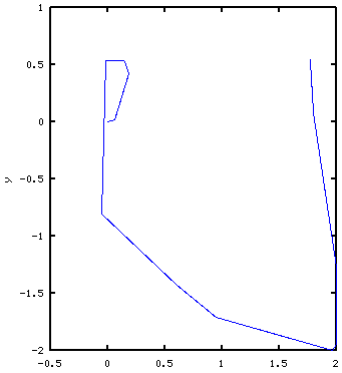


Figura 4.9: Projeção de uma rota calculada em $SE(2)$ para uma superfície em $\mathbb{R}^2$

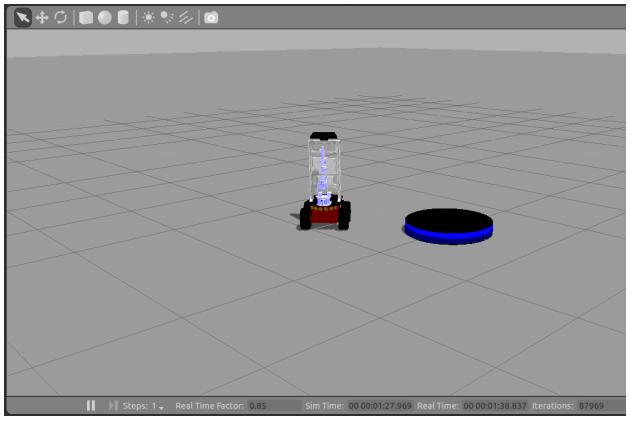
calculadas algumas juntas podem responder irregularmente, como na figura 4.8, onde a junta *Elbow Pitch* mostra oscilações entre a posição atual e a posição desejada da junta apesar de conseguir chegar a posição final corretamente.

4.2 Execução de trajetórias para a base móvel

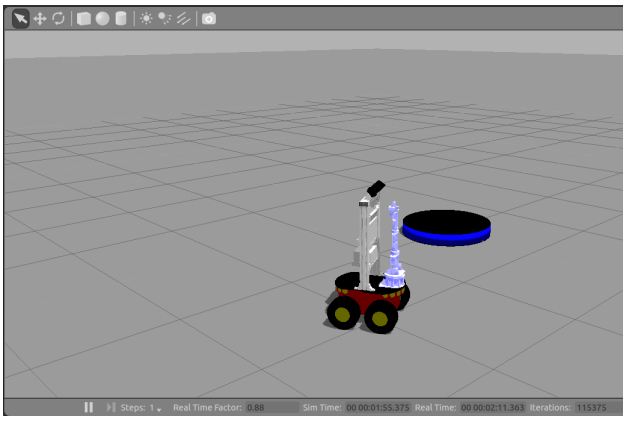
Para testar o módulo de planejamento e execução de trajetórias foi colocado um cilindro de 40 cm de diâmetro, na posição (1,0) do eixo de coordenadas inercial, e dada a tarefa ao planejador de levar o robô da configuração $q_I = (x = 0;\ y = 0;\ \theta = 0)$, até uma configuração $q_G = (1,8;\ 0.55;\ \pi/2)$.

O planejador encontrou uma rota (Fig. 4.9) que levou q_I até uma configuração final desejada $(1,77708;\ 0,54067;\ 1,56466)$.

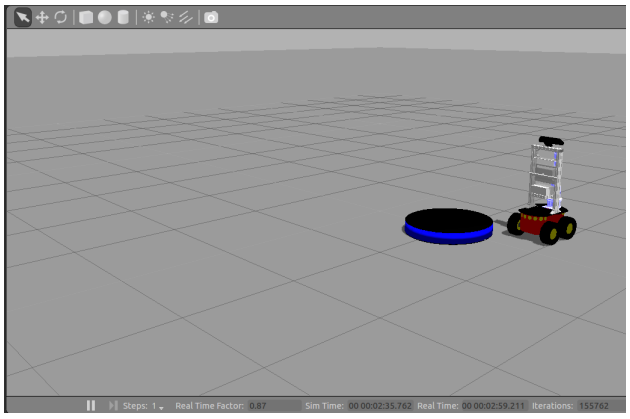
A execução da rota (Fig. 4.10) levou o robô até uma configuração final $(1,774180;\ 0,510048;\ 1,561688)$.



(a)



(b)

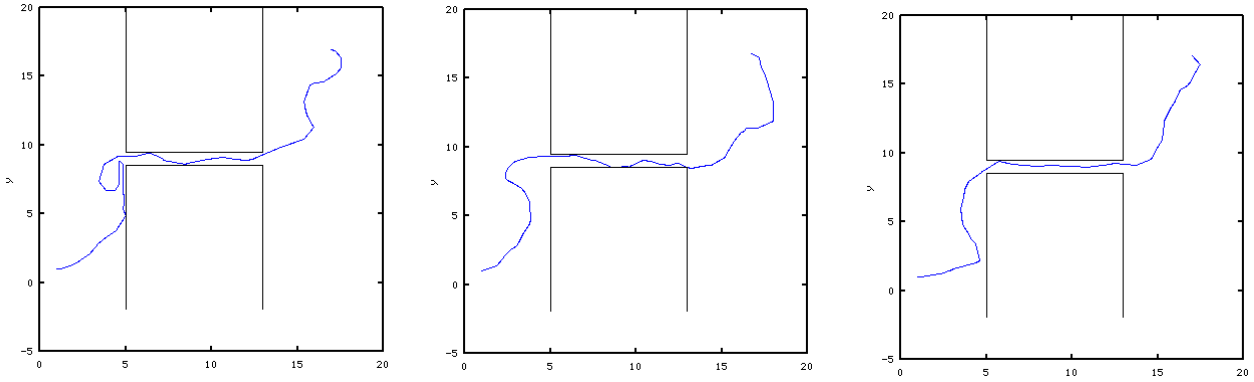


(c)

Figura 4.10: Execução de uma rota para a que a base móvel desvie de um obstáculo.

Algoritmo	Média de configurações amostradas válidas	Média de configurações na rota calculada
RRT	261809,3	62,35
EST	56849,65	80,9
K-PIECE	25271,9	128,9

Tabela 4.1: Teste dos algoritmos RRT , EST, e K-PIECE



(a) RRT. Rota com 77 configurações. (b) K-PIECE. Rota com 92 configurações (c) EST. Rota com 46 configurações

Figura 4.11: Projeções de rotas para o problema com $q_I\,(1;\,1;\,0)$ e $q_G\,(17;\,17;\,-\pi)$

4.3 Avaliação de algoritmos de planejamento

4.3.1 Base móvel

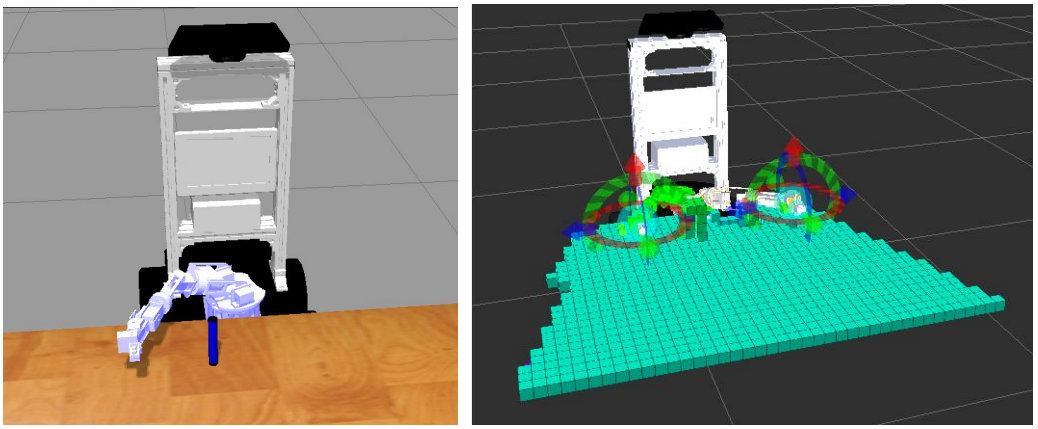
No caso da base móvel vamos levar em conta dois aspectos do planejamento: O número de configurações amostradas válidas, pois para validar uma configuração é necessário propagar a configuração atual, o que pede uma integração numérica que pode ter um custo computacional alto. E também o número de configurações dentro da rota calculada, pois a estratégia de execução de rotas possui instantes de velocidade zero para o robô quando ele está se movimentando entre configurações da rota, dessa maneira um menor número de configurações na rota calculada leva a um tempo menor de execução de uma rota para o problema de questionamento único.

São testados por 20 vezes três algoritmos, RRT, EST [19], KPIECE [20], em um ambiente com dois grandes espaços abertos e uma passagem estreita. As configurações inicial e final são $q_I\,(1;\,1;\,0)$ e $q_G\,(17;\,17;\,-\pi)$. Os resultados são mostrados na tabela 4.1 e na figura 4.11.

4.3.2 Braço robótico

Para avaliar os algoritmos de planejamento para o braço robótico mede-se a trajetória percorrida pela garra no espaço, além de quanto tempo o planejador levou para encontrar uma solução.

No planejamento para o braço robótico $\mathcal{W} = \mathbb{R}^3 \times \mathbb{H}$, ou seja, as rotações no espaço são representadas por quatérnios. Ambas configurações final e inicial possuem o mesmo quatérnio



(a) *Porthos* logo antes da execução do teste (b) Configurações inicial e final do braço representadas no *MoveIt!*

Figura 4.12: Problema de planejamento de rota para o braço robótico *cyton I*.

unitário para orientação da garra, $h_1 = -0,5i - 0,5j + 0,5k - 0,5$. As posições inicial e final da garra são dadas por:

- $P_I : x = 4,7272;\ y = -0,1687\,;\ z = 0,46282;$
- $P_G : x = 4,4697;\ y = 0,3117\,;\ z = 0,46259;$

É importante observar que o *MoveIt!* possui um módulo de cinemática inversa que transforma a posição e a orientação da garra em uma configuração $q \in \mathbb{R}^7$ para o braço robótico.

No teste, também foi colocada uma peça cilíndrica entre as duas configurações para que o algoritmo calculasse uma rota que a contornasse (Fig. 4.12).

O *MoveIt!* utilizou-se do *OctoMap* construído a partir da nuvem de pontos do kinect para checar as colisões entre o braço e a peça.

O *OctoMap* foi construído com resolução de 3 cm.

Os algoritmos usados para planejamento foram o EST, K-PIECE e RRT em versões que não usam restrições cinedinâmicas, além dos algoritmos RRT-Connect e PRM.

Para cada algoritmo, o planejador é executado 20 vezes.

As medidas estão na tabela 4.2.

Algoritmo	Percurso médio da garra (m)	Tempo médio para cálculo de rota (s)
RRT	1,07	1,34
RRT-Connect	0,6968	0,09
PRM	1,0548	0,25
K-PIECE	1,2727	0,36
EST	1,5225	0,58

Tabela 4.2: Teste dos algoritmos EST, KPIECE, PRM, RRT-Connect, e RRT para o braço robótico.

Capítulo 5

Conclusões

Ao longo desse manuscrito foram apresentados fundamentos teóricos de planejamento de rotas que serviram como base para propor uma arquitetura de *software* para planejamento de rotas (Figuras 3.22, e 3.21) para o manipulador móvel do LARA, o Porthos. Com a implementação dessa plataforma foi possível planejar tarefas de manipulação de peças pelo braço robótico do Porthos, assim como tarefas de movimento da base móvel.

A implementação das estratégias de planejamento e execução de rotas foi facilitada por iniciativas de código aberto como o ROS, MoveIt!, e Gazebo, sem as quais não teria sido possível realizar um trabalho com a portabilidade e flexibilidade necessárias para futuras extensões desse trabalho.

Os módulos de planejamento e execução apresentaram variados níveis de sucesso e eficácia. Os testes realizados no capítulo 4 mostram, por exemplo, que a estratégia de execução de rotas usada para a base móvel leva o robô para uma configuração muito próxima da configuração final desejada. No entanto, a estratégia de execução de rotas utilizada leva a uma sequência de velocidades descontínuas do robô. Isso porque a velocidade linear pode variar de zero a V_{max} quando o robô está se movendo de um ponto a outro da rota. Essa abordagem conservadora é interessante para garantir que o seguimento da rota seja cumprido, mas claramente não é o ideal pois poderia haver uma maneira de controlar as velocidades linear e angular do robô que garantisse o cumprimento da trajetória levando o robô à parada completa somente na sua configuração final e não mais em todos os pontos intermediários.

Ainda sobre a base móvel, o planejamento de rotas foi testado com três algoritmos, RRT, K-PIECE, e EST, para os quais foi possível perceber que o K-PIECE amostrou menos configurações antes de encontrar uma solução, sendo portanto mais rápido que os outros, mas apesar disso possuiu o maior número de pontos na rota calculada, o que se traduz em uma maior lentidão na execução da rota. Por esse motivo, são preferidos os outros dois algoritmos ao K-PIECE para planejamento de rotas para a direção diferencial do Porthos, sendo que o RRT possui melhor performance na execução apesar de demorar um pouco mais para calcular uma rota.

Para o braço robótico foi possível perceber que os controladores PID levam a um seguimento da rota satisfatório, a menos da junta *Shoulder Pitch* onde não foi possível retirar o sobrepasso ao executar trajetórias para a junta.

Dos testes em planejamento de rotas para o braço robótico é possível perceber claramente uma melhor performance do algoritmo RRT-Connect em relação aos outros. O RRT-Connect gerou percursos menores para a garra, além de ter calculado um plano de rotas para todas as juntas em tempo menor do que os outros algoritmos testados.

A maior contribuição desse trabalho foi na integração de ferramentas computacionais para uma simulação do robô. A partir disso é possível estender o projeto de várias maneiras, dentre outros motivos, por causa da natureza de código aberto da maioria das ferramentas utilizadas. Não obstante, o projeto mostra uma maneira extremamente escalável para outros robôs de como simular planejamento e execução de rotas no contexto da robótica móvel.

Ainda existem vários pontos a se melhorar e estender o trabalho, principalmente na execução das rotas calculadas pelos planejadores. Os módulos de execução de rotas, tanto da base móvel quanto do braço robótico, necessitam de melhorias para que executem as rotas de maneira mais suave e precisa.

Outro ponto importante para extensão do trabalho é o planejamento de corpo inteiro para o robô, onde o espaço de configurações do robô consistiria não mais de espaços distintos, $\mathcal{C}_{base}$, $\mathcal{C}_{arm}$, mas sim de um grande espaço $\mathcal{C}_{base} \times \mathcal{C}_{arm}$ onde o planejador faria um caminho que respeitasse não somente o planejamento do braço, como também o planejamento da base móvel e suas restrições cinedinâmicas. Em [21], é feito um trabalho parecido com essa premissa, não com robôs manipuladores, mas sim com um espaço de configurações $\mathcal{C}_{base} \times \mathcal{C}_{arm}$ idêntico ao caso do Porthos, sendo uma clara sugestão para começar um trabalho futuro. Essa extensão provavelmente consistirá em uma contribuição ao código aberto do MoveIt!.

Por causa dos vários módulos para planejamento de rotas para robôs já existentes no MoveIt! seria possível, por exemplo, usar *OctoMaps* para gerar grades de ocupação que seriam usadas, não somente para o manipulador, como também para a base móvel.

A possibilidade de extensão do trabalho, inclusive para robôs reais, não somente simulados, e o sucesso obtido até agora na integração das várias ferramentas computacionais, permitem concluir que o objetivo inicial de desenvolver uma plataforma para simulação do Porthos foi realizado com sucesso.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] SIEGWART, R.; NOURBAKHSH, I.; SCARAMUZZA, D. *Introduction to Autonomous Mobile Robots*. Mit Press, 2011. (Intelligent Robotics and Autonomous Agents). ISBN 9780262015356. Disponível em: <http://books.google.com.br/books?id=ssp2cgAACAAJ>.

[2] LATOMBE, J.-C. *Motion Planning*. 2009. [Notas de Aula] - Disponível em <`http://ai.stanford.edu/~latombe/cs326/2009/index.htm`> - Acessado em 02/07/2014.

[3] LAVALLE, S. M. *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006. Disponível em <`http://planning.cs.uiuc.edu/`> - Acessado em 02/07/2014.

[4] HORNUNG, A. et al. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 2013. Disponível em: <http://octomap.github.com>.

[5] KOZLOWSKI, K.; DUTKIEWICZ, P.; WRÓBLEWSKI, W. *Modeling and Control of Robots*. Warsaw, Poland: Wydawnictwo Naukowe PWN, 2003. In Polish.

[6] LAVALLE, S. M. Motion planning. *Robotics & Automation Magazine, IEEE*, IEEE, v. 18, n. 1, p. 79–89, 2011.

[7] QUIGLEY, M. et al. ROS: an open-source robot operating system. *ICRA workshop on open source software*, v. 3, n. 3.2, p. 5, 2009.

[8] MARDER-EPPSTEIN, E. et al. The office marathon: Robust navigation in an indoor office environment. In: *International Conference on Robotics and Automation*. [S.l.: s.n.], 2010.

[9] LUCA, A. D.; ORIOLO, G.; SAMSON, C. Feedback control of a nonholonomic car-like robot. In: LAUMOND, J.-P. (Ed.). *Robot Motion Planning and Control*. Berlin: Springer-Verlag, 1998. p. 171–253.

[10] KAVRAKI, L. E. et al. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics & Automation*, v. 12, n. 4, p. 566–580, jun. 1996.

[11] BARRAQUAND, J. et al. A random sampling scheme for robot path planning. In: GIRALT, G.; HIRZINGER, G. (Ed.). *Proceedings International Symposium on Robotics Research*. New York: Springer-Verlag, 1996. p. 249–264.

[12] ADORNO, B. *Planejamento probabilístico de rotas no espaço de configuração e sua aplicação em robótica móvel*. Brasília, DF, 2008. Dissertação de mestrado em Engenharia Elétrica, Universidade de Brasília.

[13] KOENIG, N.; HOWARD, A. Design and use paradigms for gazebo, an open-source multi-robot simulator. v. 3, p. 2149–2154.

[14] ŞUCAN, I. A.; CHITTA, S. MoveIt! 2012. `http://moveit.ros.org/`.

[15] ŞUCAN, I. A.; MOLL, M.; KAVRAKI, L. E. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, v. 19, n. 4, p. 72–82, December 2012. `http://ompl.kavrakilab.org`.

[16] MEAGHER, D. Geometric modeling using octree encoding. *Computer Graphics and Image Processing*, v. 19, n. 2, p. 129 – 147, 1982. ISSN 0146-664X. Disponível em: <http://www.sciencedirect.com/science/article/pii/0146664X82901046>.

[17] BOYCE, W. E.; DIPRIMA, R. C. *Equações Diferenciais Elementares e Problemas de Valores de Contorno*. Travessa do Ouvidor,11 Rio de Janeiro-RJ,Brasil: LTC, 2006. ISBN 975-85-216-1499-9.

[18] KUFFNER, J.; LAVALLE, S. RRT-connect: An efficient approach to single-query path planning. v. 2, p. 995–1001 vol.2, 2000. ISSN 1050-4729.

[19] HSU, D.; LATOMBE, J.-C.; MOTWANI, R. Path planning in expansive configuration spaces. In: *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*. [S.l.: s.n.], 1997. v. 3, p. 2719–2726 vol.3.

[20] ŞUCAN, I.; KAVRAKI, L. A sampling-based tree planner for systems with complex dynamics. *Robotics, IEEE Transactions on*, v. 28, n. 1, p. 116–131, Feb 2012. ISSN 1552-3098.

[21] LIN, Y. et al. Lift path planning for a nonholonomic crawler crane. *Automation in Construction*, v. 44, n. 0, p. 12 – 24, 2014. ISSN 0926-5805. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0926580514000594>.

ANEXOS

I. DESENHOS TÉCNICOS PARA O PORTHOS

O Porthos é formado por um robô *Pioneer* modelo P3-AT, um braço robótico *Cyton I*, e periféricos e acessórios que são montados em cima do *pioneer*, ou em um suporte.

Tanto o suporte quanto o braço robótico não possuíam representações em 3D que pudessem ser usadas nas simulações e descrições do robô usadas nesse trabalho, por isso, foi necessário desenhá-los em um ambiente CAD.

Os desenhos técnicos do suporte e do braço robótico encontram-se nas figuras seguintes assim como uma ilustração dos respectivos modelos URDF.

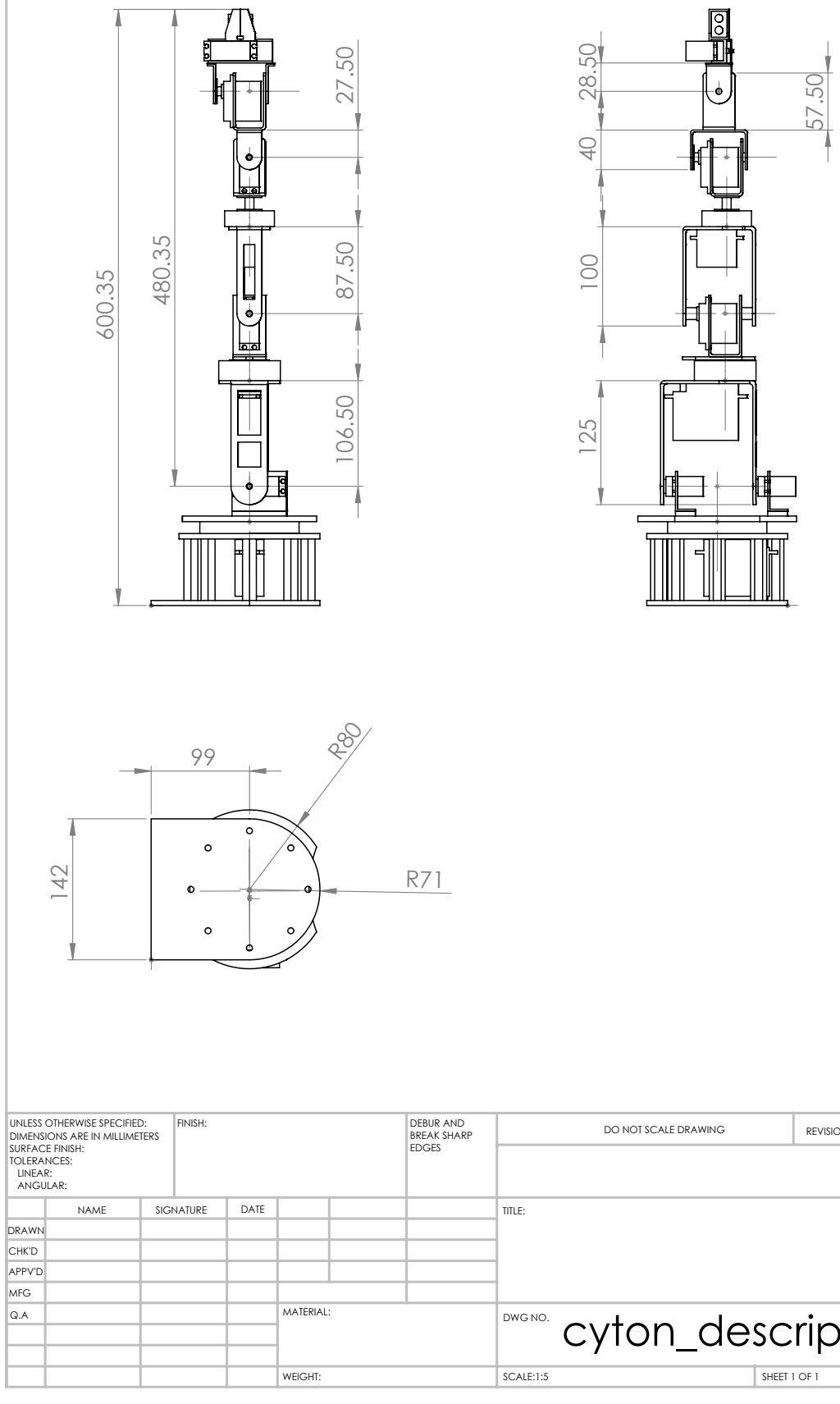


Figura I.1: Desenho do *Cyton I*

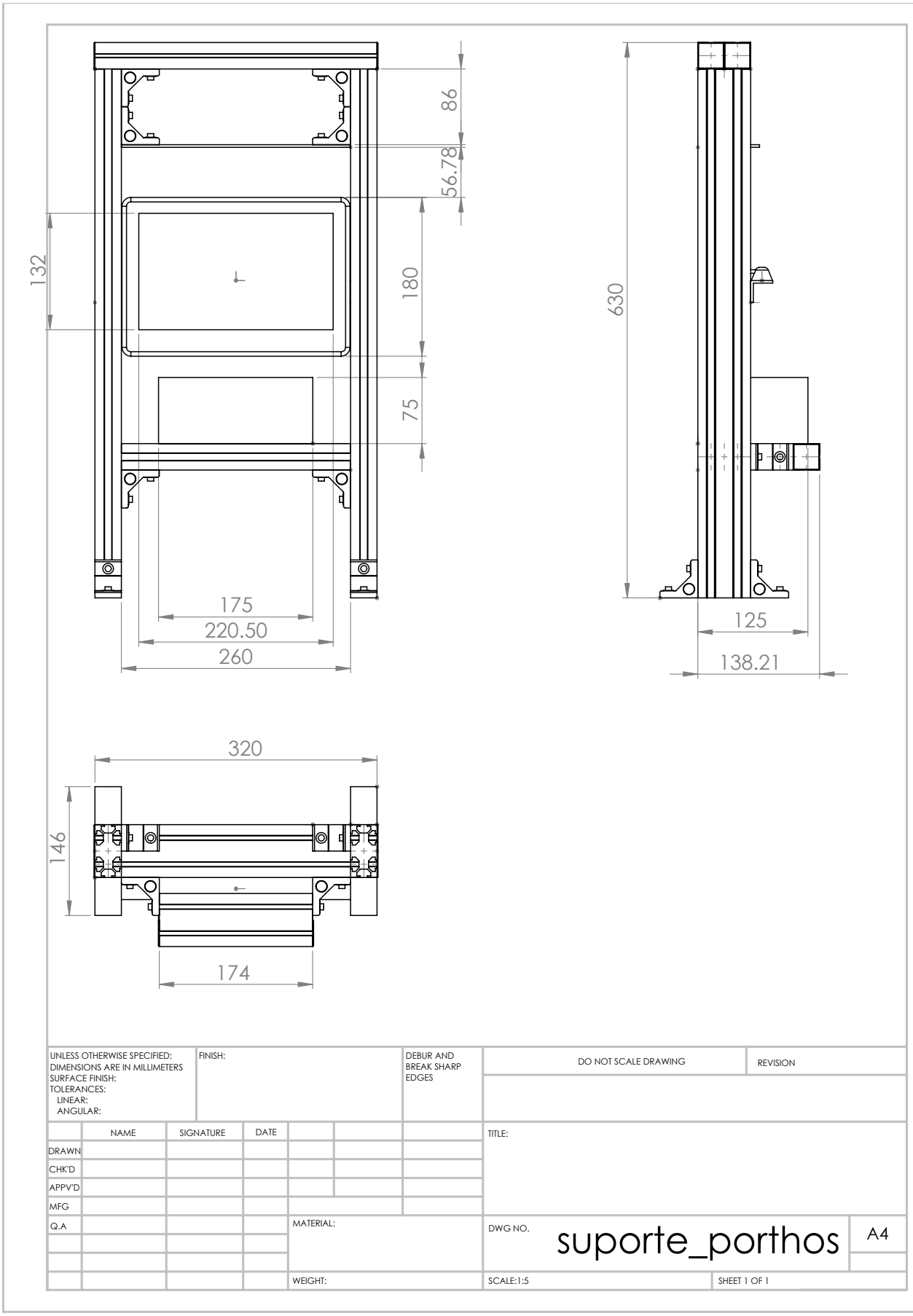


Figura I.2: Desenho do suporte usado sobre o *Porthos*

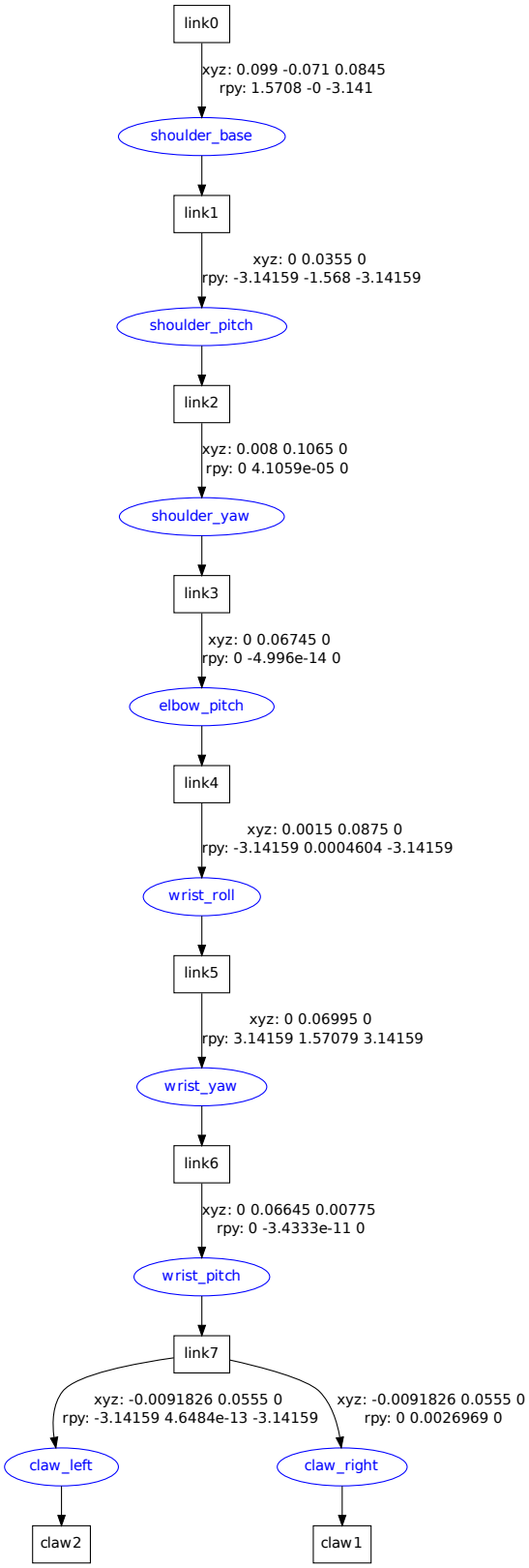


Figura I.3: Representação em árvore do modelo URDF do *cyton I*

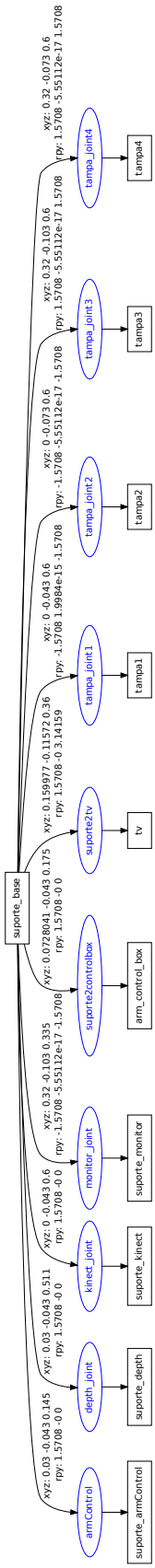


Figura I.4: Representação em árvore do modelo URDF do suporte sobre o Porthos

II. DESCRIÇÃO DO CONTEÚDO DO CD

O CD segue a seguinte estrutura:

Ele possui o relatório e o resumo do relatório em PDF e possui três pastas que seguem a seguinte estrutura:

- apresentacao/: A apresentação utilizada e os vídeos utilizados na apresentação.
- programas/: Programas utilizados nesse trabalho.
- referencias/: Referências bibliográficas utilizadas nesse trabalho.

III. PROGRAMAS UTILIZADOS

Foram utilizados os seguintes programas nesse trabalho:

- Escrita do relatório:
  - Usou-se o Lyx como ambiente de desenvolvimento para criar o texto usando TEX.
  - Usou-se o inkscape e o KolourPaint para o desenvolvimento de figuras.
  - Para os gráficos utilizou-se o *Octave* e o *rqt_plot*, uma ferramenta nativa do ROS.
- Simulações
  - Gazebo: Uma ferramenta computacional para simulação de vários robôs.
- Algumas bibliotecas utilizadas
  - OMPL: Biblioteca para planejamento de rotas que usa algoritmos de planejamento baseados em amostragem do espaço de configurações;
  - ROS: *Framework* utilizado para integrar vários módulos necessários em um sistema computacional feito para robôs.;
  - MoveIt!: Pacote do ROS usado para planejamento e execução de trajetórias de um robô.