UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
DEPARTAMENTO DE INFORMÁTICA
MESTRADO EM INFORMÁTICA

UM AMBIENTE PARA SIMULAÇÃO DE DINÂMICA E CONTROLE DE ROBÔS COM ANIMAÇÃO EM 3D

ALESSANDRA AGUIAR VILARINHO

VITÓRIA
2003

HANS-JORG ANDREAS SCHNEEBELI

SIMULAÇÃO COMPUTACIONAL DE DINÂMICA E CONTROLE DE ROBÔS COM ANIMAÇÃO 3D

Dissertação apresentada ao Mestrado em Informática da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Mestre em Informática.
Orientador: Prof. Dr. Hans-Jorg Andreas Schneebeli.

VITÓRIA
2003

ALESSANDRA AGUIAR VILARINHO

SIMULAÇÃO COMPUTACIONAL DE DINÂMICA E CONTROLE DE ROBÔS COM ANIMAÇÃO 3D

COMISSÃO EXAMINADORA

_____

Prof. Hans-Jorg Andreas Schneebeli, D. Sc.
Orientador

_____

Chin Ting Wu
Prof[a.], D. Sc.

_____

Alberto Ferreira de Souza
Prof. , D. Sc.

_____

Teodiano Freire Bastos Filho
Prof. , D. Sc.

Vitória, outubro de 2003.

Dedicatória

Dedico este trabalho à minha irmã, pois sua vitória ofusca todo o brilho de minha conquista.

Agradecimentos

Agradeço primeiramente as pessoas mais próximas, que tiveram que entender minha ausência, distância e nervosismo durantes as fases finais de preparação deste documento. Agradeço em especial aos meus pais, Arlete e Guarino, pois sem seus esforços, que não foram poucos, não teria chegado até aqui. Agradeço também ao meu marido, Rodrigo, por me ajudar nos momentos mais difíceis quando pensei em até mesmo desistir. Agradeço também às amigas Joelma, Renata e Vanessa, que me fizeram o favor de ler meu texto ainda nas fases preliminares, um verdadeiro rascunho de dissertação.

Li no livro "Como se faz uma tese", do Umberto Eco, que não se deve agradecer em demasia ao orientador, pois ele não fez mais do que a sua obrigação. Porém não penso desta forma, acho que devemos ter gratidão às pessoas que nos ajudam, principalmente quando elas fazem mais do que suas obrigações, e foi o que o meu orientador fez. Além de sua orientação para a realização do trabalho, ele teve muita paciência e não deixou que eu desanimasse em nenhum momento. Também não posso esquecer dos professores do Mestrado em Informática com os quais tive as aulas para cumprimento de créditos: Crediné, Dede, Ricardo e Saulo.

Não posso deixar de agradecer aos amigos do LAI (Laboratório de Automação Inteligente do departamento de Engenharia Elétrica da UFES): Eliete, Érico, Josemar, Danilo, Rennow e Rodrigo. Tivemos muitos momentos divertidos em meio à aflição de nossos trabalhos de mestrado e doutorado.

Por último agradeço a Deus por ter colocado em minha vida todas estas pessoa, as quais devo enorme gratidão, pois me ajudaram em mais uma etapa da minha vida.

"Maravilhoso é volver os olhos para trás e constatar quantos obstáculos vencidos, quantos sacrifícios, quantos esforços, quantas preocupações ...

Mais maravilhoso ainda é olhar para frente com fé, sabendo que existe uma força maior, que nos acompanha dia-a-dia e que ao descortinarmos um novo horizonte poderemos fazer o bem, doando àqueles que precisam, um pouco do que somos e sabemos."

Autor Desconhecido

Lista de Tabelas

Lista de Figuras

Sumário

Resumo

Este trabalho mostra a utilização da teoria de dinâmica de corpos rígidos e da teoria clássica de controle para a execução da simulação computacional de robôs. Mostra também a utilização da computação gráfica para a apresentação dos resultados da simulação através de uma animação em três dimensões do modelo geométrico do robô simulado. Este trabalho faz uso de teorias e ferramentas da física, matemática, automação e computação para a solução de um problema em robótica: simulação computacional de controle e dinâmica de robôs com apresentação dos resultados através de uma animação gráfica em três dimensões.

Abstract

This work shows the use of the rigid body dynamic's theory and the classic control's theory as a base for simulation of robots. The simulation's results are presented in an animation, in three dimensions, of the geometric model of the robot. This work uses theories and tools of Physics, Mathematics, Automation and Computation to solve a robotic's problem: control and dynamic's robot computational simulation with three dimensions animation.

1 Introdução

A computação gráfica tem sido usada cada vez mais como ferramenta de apoio para simuladores. Uma simulação computacional pode ter seus resultados apresentados de diversas formas tais como: tabelas, gráficos ou listagem com os valores obtidos. Entretanto nada disso se compara ao poder de uma apresentação, na qual é possível a visualização de objetos de maneira animada. Uma animação em três dimensões, também chamada de animação 3D, pode facilitar a interpretação dos dados em análise.

Em especial, a simulação computacional aplicada em robótica é uma ferramenta imprescindível às fases de projeto, construção e teste de um robô. Especificamente, testar o funcionamento de um robô através de uma simulação computacional é mais rápido, menos custoso e mais seguro, do que fazer testes em campo, pois muitas vezes pode-se colocar em risco o equipamento e tudo mais que o cerca, até mesmo, pessoas [ADADE]. É muito mais fácil analisar a movimentação de um robô através da animação de seu modelo geométrico do que através de gráficos que demonstram como, por exemplo, as variáveis posição (translação e rotação), velocidade, força e torque se relacionam entre si ao longo do tempo.

Existem diversos trabalhos de simulação que apresentam os resultados através de uma animação, considerando apenas os aspectos cinemáticos durante a determinação do movimento do robô. O foco deste trabalho é a realização de uma simulação computacional de robôs levando-se em consideração: **os comandos de movimentação** passados ao robô, **as leis de controle** que regem a movimentação do robô e **as equações de dinâmicas** que descrevem o comportamento dinâmico do mesmo.

A execução da simulação é um ciclo no qual verifica-se a existência de um comando de movimentação; calcula-se a força/torque necessária à realização do comando, usando-se para este cálculo a lei de controle; e aplica-se esta força/torque ao modelo dinâmico do robô, o que gera a cada ciclo uma nova posição/orientação do robô. A apresentação dos resultados é feita através da animação, ou redesenho do modelo geométrico do robô em cada nova posição/orientação calculada no ciclo da simulação. É possível realizar a simulação para robôs distintos com precisão, eficiência e

flexibilidade, pois os comandos, as leis de controle, as equações de dinâmica e o modelo geométrico do robô são parametrizáveis, ou seja, são entradas para o sistema de simulação. Além disso, a teleoperação também é uma característica importante que pode ser alcançada pelo fato de comandos de movimentação fazerem parte da simulação, ou seja, a simulação também pode servir de ambiente virtual para o treinamento do usuário no comando de um robô.

1.1 Robótica

A robótica é considerada uma área multidisciplinar [CRAIG] por envolver conhecimentos de engenharia mecânica, que contribuem para o entendimento do comportamento estático e dinâmico dos robôs; por envolver conhecimentos de física, que provê o ferramental físico e matemático necessário à descrição do movimento espacial dos robôs; por envolver conhecimentos de engenharia elétrica, que contribuem com a teoria de controle necessária ao controle do movimento do robô quando o mesmo precisa realizar uma tarefa; e por envolver também a ciência da computação, que provê a base para programação dos robôs para que os mesmos executem as tarefas desejadas.

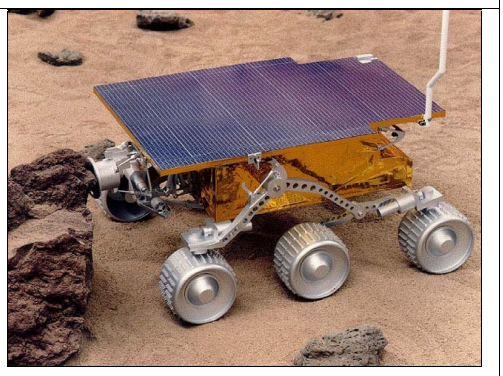
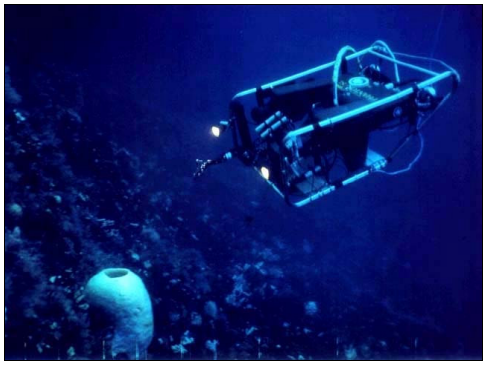
Em princípio robô é, no contexto deste trabalho, uma palavra utilizada para dar nome a um conjunto muito amplo de dispositivos mecânicos tais como: veículos terrestres ou aquáticos, braços mecânicos, andróides, e outros, todos eles controlados remotamente ou não, por pessoas ou por computadores. Podemos classificar os robôs em dois grandes grupos: quanto à forma de locomoção ou quanto a sua programação, ambas descritas a seguir.

Classificação quanto à forma de locomoção

Todos os robôs possuem alguma forma de movimentação, porém, existem aqueles que se movimentam sempre no mesmo espaço e aqueles que se movimentam livremente.

- **Robôs móveis** - quando a movimentação do robô não é limitada a um espaço dizemos que este robô é **móvel**. Os robôs móveis possuem ainda, outras subclassificações de acordo com a forma como executam sua locomoção, ver tabela da figura 1.1.
- **Robôs estacionários** - quando a movimentação do robô está restrita a um espaço limitado dizemos que este robô é **estacionário**, ou seja, não se locomove. Ele pode ter partes fixas e partes móveis com movimentos limitados a um espaço de trabalho.

A tabela da figura 1.1 mostra a classificação quanto à forma de locomoção.

MOVEL	Patas	 Robô Dante da NASA [NASA]
	Veículos terrestres ou aquáticos	 Robô Sojourner utilizado em 1997 na missão PathFinder da NASA[NASA].  Robô Trov utilizado no mar da Antártica[NASA].

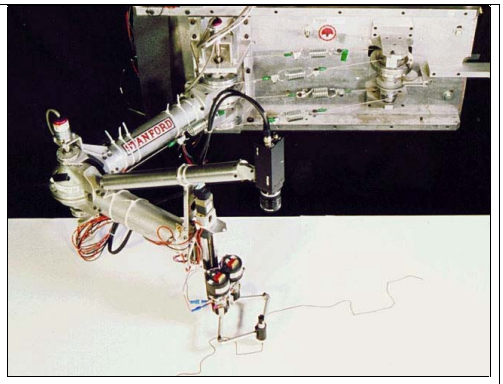
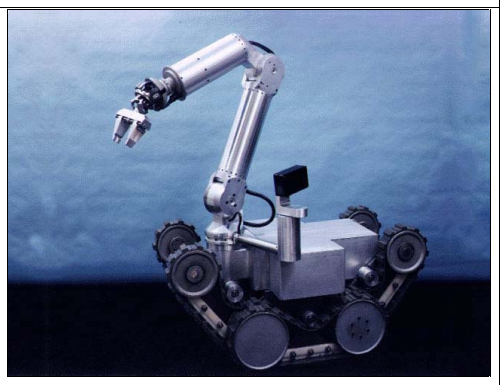
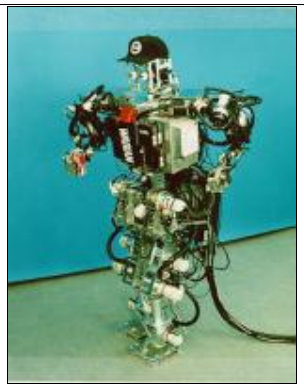
E S T A C I O N A R I O	Manipuladores	 Exemplo de robô manipulador [NASA].
M I S T O	Combinação de Veículo Terrestre e Manipulador	 Robô HAZBOT da NASA [NASA].
	Andróide	 Robô WABIAN (WAseda BIpedal humANoid) da Universidade Waseda no Japão [WABIAN].

Figura 1-1 Classificação dos robôs quanto à forma de locomoção

Classificação quanto à forma de programação.

- **Robôs autônomos** - são controlados por sistemas multifuncionais computadorizados. Têm a capacidade de interagir com o ambiente, através de sensores, e de tomar decisões em tempo real.
- **Robôs controlados por computador** - estes são semelhantes aos anteriores, porém não possuem o recurso de interagir com o ambiente.
- **Robôs de aprendizagem** - este tipo de robô limita-se à repetição de movimentos realizados com a intervenção de um operador ou memorizados em seu sistema.

Grande parte da literatura a respeito de robôs descreve detalhadamente o funcionamento dos robôs manipuladores, porém os conceitos e a análise do comportamento cinemático e dinâmico, apresentados a seguir, também podem ser estendidos para os outros tipos robôs, usaremos então os robôs manipuladores para exemplificar.

Componentes e estruturas dos robôs

Os robôs manipuladores são braços mecânicos compostos por partes conectadas umas às outras por uma ligação que permite um tipo de movimento relativo entre as partes. A ligação entre as partes é geralmente chamada de **vínculo** (*join*) e as partes são geralmente denominadas de **juntas** (*links*). Portanto um robô manipulador é um conjunto de juntas e vínculos, onde cada vínculo conecta duas juntas permitindo o movimento relativo entre elas. A mobilidade das juntas do robô depende do número e do tipo de vínculos que o mesmo possui. Existem vínculos que permitem que uma junta seja rotacionada ou transladada em relação à outra junta. Cada movimento relativo de uma junta em relação à outra representa um **grau de liberdade**. A quantidade total de movimentos relativos de todas as juntas de um robô é chamada de número de graus de liberdade de um robô.

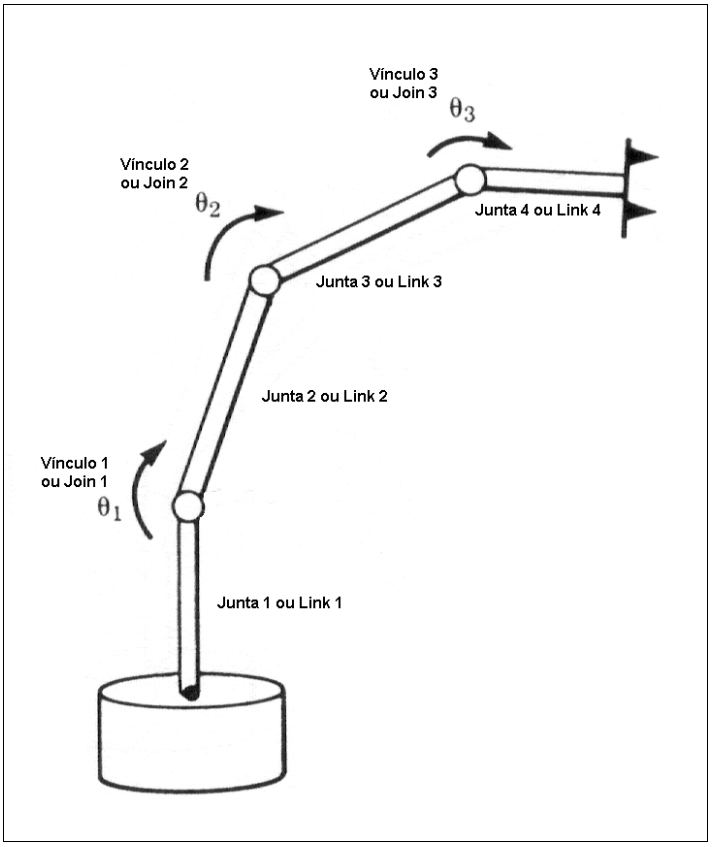


Figura 1-2 - Esquema de um robô manipulador mostrando seus vínculos e juntas

Os robôs manipuladores possuem em sua extremidade, também chamada de **atuador**, garras ou ferramentas para a execução de uma tarefa específica. É muito comum ficarem fixos em mesas ou pedestais, possuem uma base fixa e a primeira junta está presa a esta base. Cada junta possui um **acionador**[1] que é responsável pela movimentação desta junta com relação à junta anterior. Os acionadores são sistemas elétricos, mecânicos ou pneumáticos que permitem a transferência de força ou torque para as juntas para que estas se movimentem. As estruturas que vão desde a base até o vínculo imediatamente anterior ao atuador, têm a função de levar o atuador a se deslocar corretamente de acordo com a tarefa a ser realizada [PAZOS]. A trajetória a ser executada pode ser especificada em função da posição de cada junta do robô ou em função do atuador.

[1] Na literatura em Português sobre Robótica, alguns autores usam o termo atuador ou atuador final para os termos atuador e acionador usados neste trabalho.

Análise da cinemática e da dinâmica dos robôs

A análise da cinemática de corpos faz o estudo do movimento destes corpos no tempo, sem se preocupar com o que causa o movimento dos corpos. Já a análise da dinâmica de corpos, faz o estudo das forças, torques e massas que causam variações nas quantidades cinemáticas no tempo, ou seja, faz o estudo do que causa a movimentação dos corpos.

A análise da cinemática dos corpos rígidos pode ainda ser dividida em cinemática direta e cinemática inversa. No caso de robôs, de acordo com como é feito o cálculo da posição e da orientação do atuador do robô manipulador temos:

No caso chamado de **cinemática direta**, calcula-se a movimentação final do atuador a partir dos deslocamentos que cada junta do robô realiza.

No caso chamado de **cinemática inversa**, calcula-se o deslocamento que cada junta do manipulador deve realizar para que o movimento resultante seja o movimento esperado do atuador.

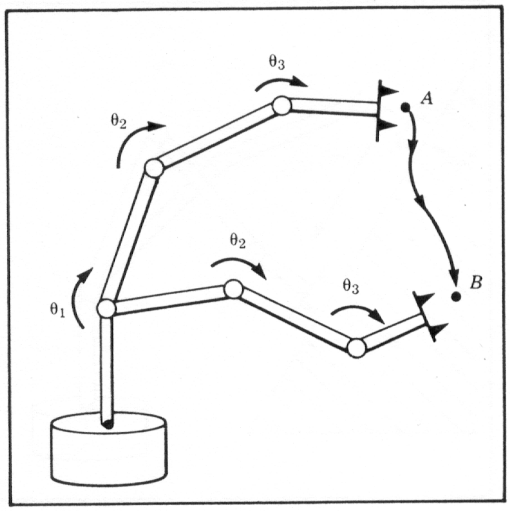


Figura 1-3 - Esquema de um robô manipulador [CRAIG]

A análise da dinâmica de copos rígidos, da mesma forma que a cinemática, também pode ser dividida em dinâmica direta e dinâmica inversa. Para o caso dos robôs temos:

Na **dinâmica direta**, calcula-se a variação das quantidades cinemáticas (posição e velocidade), a partir de forças/torques aplicadas às juntas.

Na **dinâmica inversa**, calcula-se quais devem ser as forças e os torques aplicados às juntas para que as mesmas sofram uma desejada variação das quantidades cinemáticas (posição e velocidade).

A maioria dos sistemas de simulação computacional em robótica é baseada em cinemática direta e inversa. Com este tipo de simulação, é possível visualizar a área de trabalho do robô, ou seja, todas as configurações de posição e orientação que o atuador pode assumir. A figura 1-4 mostra em (a) um robô manipulador, seguido de seu esquema simplificado em (b) e em (c) uma vista lateral e uma vista superior, onde a área hachurada representa a área de trabalho ou de atuação do manipulador.

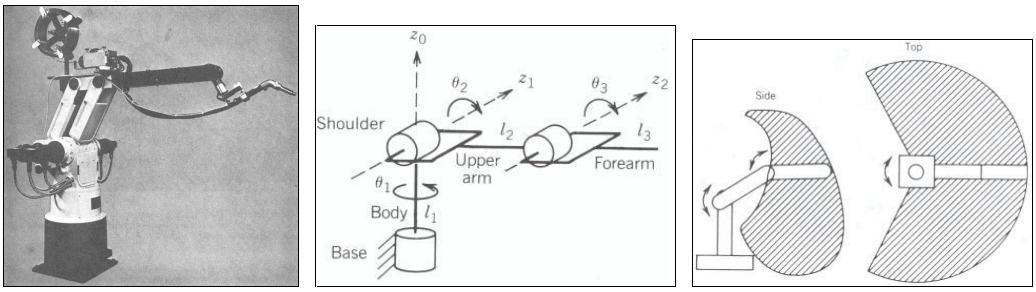


Figura 1-4 - Manipulador (a), sua representação (b) e sua área de trabalho (c) [SPONG]

A limitação destas simulações computacionais está no fato de que se fazendo somente os cálculos de cinemática, não é possível saber qual é a força/torque, também chamada de atuação, necessária para causar os deslocamentos nas estruturas do manipulador para que o atuador alcance a posição e orientação desejadas. Uma simulação que informe qual é a atuação necessária para a realização de uma determinada tarefa auxilia o projeto do robô, pois é possível saber de antemão quais são as características necessárias dos acionadores, ou seja, qual é o torque, a força e a velocidade que cada acionador deverá fornecer aos vínculos do manipulador.

Além da simplificação de tratar a movimentação do robô como um problema de cinemática, a maioria das simulações é implementada para um modelo específico de robô, podendo-se variar somente alguns parâmetros geométricos deste robô.

Uma situação ideal e desejada é implementar um sistema de simulação que determine o movimento do robô a partir das leis da física (cinemática e dinâmica) e que possua também uma generalidade para que diversos robôs sejam simulados através da

parametrização de suas características geométricas, de seus comportamentos dinâmicos e da lei de controle que atua sobre a movimentação dos mesmos. Devemos simular o sistema de controle da movimentação do robô, pois simular somente o comportamento dinâmico do robô para um dado conjunto de forças/torques aplicados ao mesmo, não é suficiente. Deve ficar claro que a intenção é simular um robô recebendo comandos de movimentação, para isto, introduzimos também um modelo de controle do robô. O modelo de controle de robô é o responsável por gerar a atuação (força ou torque) necessária para que um determinado comando de movimentação seja executado.

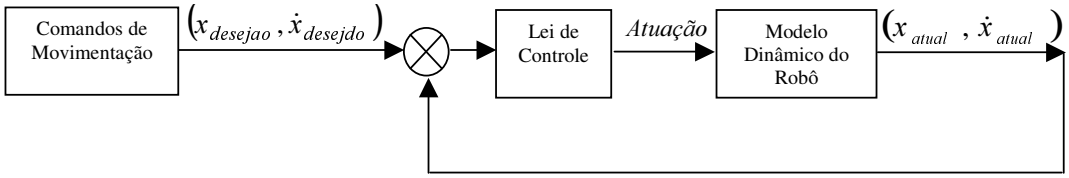


Figura 1-5 - Interação entre os Comandos de Movimentação, a Lei de Controle e o Modelo de Dinâmica do robô

1.2 Animação baseada em física

Em alguns textos a animação baseada em física também é chamada de animação baseada em dinâmica. Animação é um tema estudado na área de Computação Gráfica. Animar é o ato de passar a sensação de movimento através da superposição rápida e progressiva de "quadros" (imagens estáticas), onde cada em cada quadro os objetos são redesenhados em novas posições [SILVA F.]. Desta forma simula-se a idéia de movimento e tempo transcorrendo. Esta técnica de animação é conhecida como animação quadro a quadro ou *keyframing*. Animação baseada em física é uma animação cuja posição e orientação dos objetos que compõem o quadro são geradas através de cálculos baseados nas leis da dinâmica.

As ferramentas computacionais podem auxiliar na construção de animações basicamente de duas formas [SILVA M.], a saber:

Animação Auxiliada por Computador: tarefas repetitivas, tais como editar e colorir, são automatizadas.

Animação Modelada por Computador: são criados modelos de representação das entidades gráficas que compõem a cena. Estes modelos representam objetos

geométricos, câmeras, luzes, materiais, texturas, etc. Para este tipo de animação a movimentação dos objetos pode ser feita baseada em Cinemática ou em Dinâmica.

A animação baseada em Cinemática automatiza o trabalho do desenhista, que antes tinha que desenhar todos os quadros de uma cena e agora desenha somente alguns, os quadros principais. É o software quem cria tantos quadros intermediários quantos queira o desenhista. Estes quadros intermediários são criados pelo software através de interpolação. Esta técnica provê uma forma muito intuitiva para especificar os movimentos dos objetos, porém em alguns casos a intervenção do desenhista é muito intensa para que se consiga uma animação que apresente movimentos naturais. Por exemplo, em uma cena onde exista uma colisão entre dois corpos não é possível gerar imagens próximas da realidade física usando-se somente a interpolação para gerar a movimentação dos dois corpos durante e depois do choque. É neste momento que o desenhista intervêm para dar um ar de realismo à cena.

É exatamente esta geração de movimentos que pareçam naturais que se torna um desafio para as animações computacionais. Para que isto aconteça precisamos gerar os movimentos que representem corretamente a realidade, respeitando as leis da física tanto no campo da cinemática quanto no campo da dinâmica. Simular os efeitos da massa e da inércia dos corpos, das forças, dos torques e dos impulsos aplicados a estes corpos é o que é almejado em animação baseada em física.

Animações totalmente geradas por computador foram inicialmente baseadas nas leis da cinemática, porém, esta técnica é insatisfatória quanto à qualidade do realismo das cenas geradas. Com a animação gerada a partir das leis da dinâmica é possível alcançar tal realismo, contudo o custo computacional é alto e a modelagem física de alguns efeitos é muito difícil. Animações que representem corretamente efeitos de dinâmica têm uma dificuldade inerente ao próprio problema em foco, as leis da Dinâmica dos Corpos.

Os grandes estúdios de cinema tentaram resolver estes problemas contando com a qualidade artística de seus desenhistas e projetistas de cenas. Várias técnicas para melhorar o realismo das animações foram utilizadas, podemos destacar a animação por movimento capturado [SILVA F.], que se aplica perfeitamente na produção de filmes. Porém quando falamos de animações que suportam interatividade, por exemplo, jogos, simulações e aplicações de realidade virtual, estas técnicas também não trazem bons resultados.

A animação baseada em física teve como ponto de partida a área de entretenimento. Os jogos de computador são as aplicações mais visadas em termos de investimentos. Estes tipos de aplicações possuem grande número de usuários, o que possibilita um preço final menor com a venda garantida das mesmas. Apesar de todo o investimento, as aplicações da área de entretenimento ainda não possuem realismo de qualidade. Nem sempre simulam as leis da física corretamente, fazendo apenas o uso de efeitos artificiais, não conquistando o público. Este tipo de animação computacional está deixando de ser aplicada essencialmente em entretenimento e está sendo utilizada também em simulações industriais, modelagem de escoamento de fluidos e gases, modelagem climática, modelagem de populações, modelagem de ecossistemas [HECKER] e outros. A animação está servindo como ferramenta de visualização destas simulações.

1.3 Simulação em robótica

A simulação na área de robótica é uma ferramenta muito importante para análise e projeto de mecanismos robóticos, seus correspondentes sistemas de controle e também no desenvolvimento de aplicações em robótica. Robôs possuem um alto custo de desenvolvimento e necessitam de uma forma barata e segura para a execução de testes de funcionamento. A simulação computacional provê esta forma. A simulação computacional com visualização gráfica em três dimensões (animação) provê a interpretação quase que imediata dos testes de funcionamento de um robô, podendo-se criar um ambiente de teste ou treinamento virtual para o usuário.

A simulação permite a análise e o teste de diferentes alternativas de funcionamento de um sistema, através da variação diversos parâmetros, como por exemplo, os de controle. Com a simulação, gargalos e limitações de capacidade podem ser revelados ainda durante a fase de concepção e projeto. Soluções alternativas podem ser geradas, testadas e certificadas possibilitando a verificação do desempenho da instalação muito antes de sua implementação física.

Há algum tempo atrás a execução de uma simulação gráfica só era possível em plataformas específicas e de alto custo, hoje ela já é possível em computadores pessoais o que reduziu muito custo e ampliou sua escala de utilização. Mesmo com a diminuição

dos requisitos de hardware para tais aplicações, o custo das ferramentas profissionais disponíveis no mercado ainda é alto e a interface gráfica destas ferramentas é baseada em CAD e geralmente possui alta curva de aprendizagem. Um exemplo disto é o software ADAMS® [MSC ADAMS] da empresa Mechanical Dynamics, fundada em 1977, que é uma pioneira em simulação de sistemas mecânicos. Este software permite que profissionais de engenharia construam e testem protótipos de sistemas mecânicos no computador, simulem realisticamente um comportamento operacional de movimentação, refinem e otimizem a performance do sistema, o que reduz os custos, o tempo, os riscos no desenvolvimento de produtos e aumenta ou garante um certo nível de qualidade. Este software é usado em simulações automotivas, aeroespaciais, entre outras.

Foram estes os fatores que nos guiaram na direção de implementar uma aplicação de baixo custo que apresente graficamente, através de uma animação em 3D, os resultados da simulação da movimentação de um robô qualquer que receba os comandos de movimentação do usuário, sendo que o cálculo dos movimentos do robô é feito levando-se em conta as características de dinâmica e de controle do mesmo.

1.4 Definição do problema

A intenção deste trabalho é desenvolver um sistema de simulação que receba **comandos** enviados pelo usuário ao robô, que leve em conta a **dinâmica** e o **controle** do robô no cálculo de sua movimentação e que apresente os resultados através da animação em 3D do modelo geométrico simplificado do robô, sendo que esta simulação pode ser executada para **robôs distintos**.

Quando analisamos a dinâmica de um corpo podemos utilizar uma das duas formulações mais conhecidas na literatura [SPONG].

- **Lagrange-Euler -** a análise da dinâmica do corpo é feita baseada no conceito de energia.
- **Newton-Euler** - a partir de forças externas aplicadas ao corpo e de forças de ação e reação internas do corpo, calcula-se a força total que atua no mesmo. De posse desta força total e da distribuição de massa do corpo, extrai-se o valor da aceleração. Uma vez calculada a aceleração do corpo, obtém-se a

posição e a velocidade a partir das equações de movimento de Newton. A análise do comportamento dinâmico é feita vetorialmente.

Baseados nestas formulações podemos modelar a dinâmica de um corpo de acordo com as seguintes abordagens:

- **Abordagem fora de linha (off-line),** onde a avaliação do comportamento dinâmico do corpo é feita através de equações diferenciais explícitas obtidas segundo as formulações anteriormente citadas. Em outras palavras, o comportamento dinâmico do corpo é descrito por uma ou mais equações diferenciais, geralmente não lineares.
- **Abordagem em linha (on-line),** onde a avaliação do comportamento dinâmico do corpo é feita através de cálculos a partir das forças aplicadas aos corpos e dos tipos de restrições de movimento existentes entre os corpos. A intensidade, direção e sentido de todas as forças aplicadas ao corpo assim como de suas respectivas reações são contabilizadas para a determinação da movimentação do corpo. Nesta situação não existe explicitamente uma equação que descreva o comportamento dinâmico do corpo. O cálculo da movimentação do corpo é feito baseado no tipo de interação/restrição que o corpo possui consigo mesmo e com o ambiente no qual está inserido.

Em seu trabalho Bart [BARENBURG] utilizou a abordagem on-line para desenvolver uma biblioteca, orientada a objetos em C++, a partir da qual é possível fazer uma simulação baseada em dinâmica sem a necessidade das equações explícitas de dinâmica dos corpos simulados.

No presente trabalho, por motivos de eficiência, usa-se a abordagem off-line, com a equação de dinâmica sendo fornecida pelo usuário do sistema de simulação segundo a formulação Lagrange-Euler. O controle da movimentação do robô está baseado na teoria de controle clássico sendo que a equação de controle também deverá ser fornecida pelo usuário do sistema de simulação.

O problema se resume, então, em criar um sistema de simulação de robôs que simule o recebimento de comandos de movimentação e simule a movimentação do robô de acordo com a sua dinâmica e de acordo com a lei de controle utilizada para reger sua movimentação. A figura 1-6 mostra como podemos dividir o problema em subproblemas. O **Simulador** é responsável por receber os **Comandos** para o **Robô** e

por gerar a **Animação** de acordo com as características **Geométrica**, **Dinâmica** e de **Controle** do **Robô**.

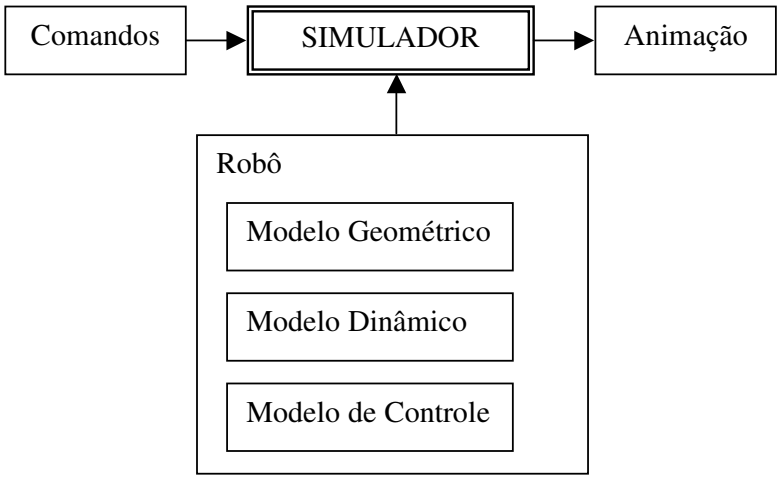


Figura 1-6 - Sub-divisão do problema

Para simular o recebimento de comandos, o robô poderá receber instruções por meio de um programa ou arquivo armazenado ou ainda diretamente através de um sistema de teleoperação (operação à distância). A entrada de dados deve ser tratada de maneira que se traduza em movimentos que os sub componentes do robô devem efetuar para que um comando, designado pelo usuário, seja realizado pelo robô. De posse desta entrada tratada, o sistema que simula o controle do robô deve ser capaz de calcular a atuação necessária ao robô para que este se movimente corretamente. A atuação calculada pelo sistema de controle é passada como entrada ao sistema que simula a dinâmica do robô, para que este, de posse da equação de dinâmica do robô, possa calcular o novo vetor de estado (posição e velocidade) do robô. O novo vetor de estados é utilizado para comunicar ao sistema de visualização as novas posições das partes do robô. O sistema de visualização se encarrega de animar a apresentação, redesenhado o robô em sua nova posição.

O sistema de controle calcula a atuação (torques/forças) que deve ser aplicada ao robô, segundo uma lei de controle. Esta lei de controle é uma função da posição e da velocidade desejada e atual do robô, ou seja, é uma função do estado desejado e do estado atual do robô. A figura 1-7 ilustra a interação entre os comandos passados ao robô, entre o sistema de controle que atua sobre sua movimentação, entre o seu modelo dinâmico e entre a animação.

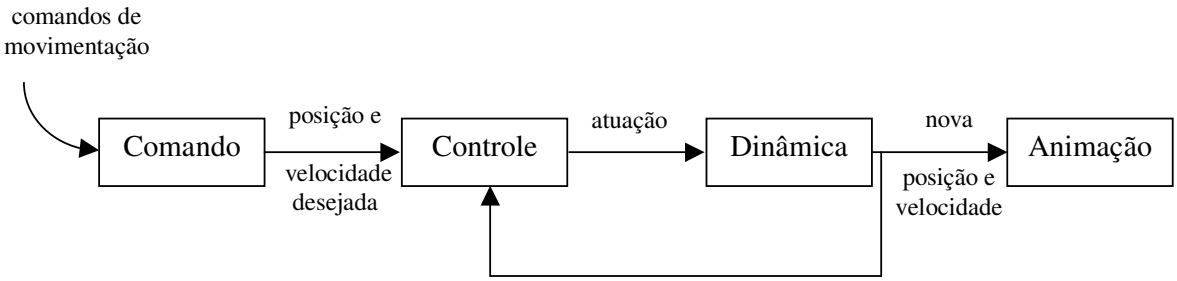
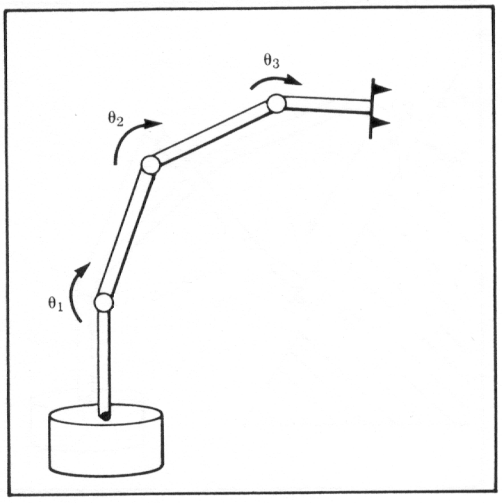


Figura 1-7 - Interação entre os módulos de Comando, de Controle, de Dinâmica e de Animação

O módulo de comando recebe os comandos de movimentação e proporciona as posições e velocidades desejadas para cada parte do robô. O módulo de controle calcula a atuação (torque/força) que deve ser passada ao módulo de dinâmica para que o robô efetue o movimento solicitado, este cálculo é feito em função da posição e velocidade desejada e atual. O modelo de dinâmica responde à atuação fornecendo novas posições e velocidades, que são consideradas posição e velocidade atual para a próxima iteração.

No caso do robô manipulador, o estado do robô (posição e velocidade) é um conjunto de *n* estados, onde *n* indica o número de graus de liberdade do robô. Este conjunto de estados é comumente chamado de vetor de estados. A figura 1-8 mostra o esquema de manipulador com três juntas rotacionais no plano XY. Neste caso o número de graus de liberdade deste manipulador é três, por tanto o seu vetor de estados ***Y*** pode ser representado por um vetor com ***2n*** elementos como mostra a figura 1-8.



$$Y = \begin{pmatrix} \theta_1 \\ \dot{\theta}_1 \\ \theta_2 \\ \dot{\theta}_2 \\ \theta_3 \\ \dot{\theta}_3 \end{pmatrix}$$

Figura 1-8 Esquema de um robô manipulador com três graus de liberdade no plano XY e seu vetor de estados Y

Além dos comandos e da lei de controle de movimentação, também será necessário conhecermos a dinâmica do robô, segundo [CRAIG], um robô manipulador tem o seu comportamento dinâmico expresso por um conjunto de equações diferenciais não lineares da forma:

$$\mathrm{T} = M(x)\ddot{x} + B(x)\dot{x}\dot{x} + C(x)\dot{x}^2 + G(x) \qquad \textbf{Equação 1-1}$$

A equação 1-1 relaciona o torque/força T aplicado aos motores com a movimentação x, onde x é uma coordenada generalizada podendo representar tanto um deslocamento linear quanto um deslocamento angular.

A animação será gerada através dos cálculos realizados pelo módulo de dinâmica. O módulo de dinâmica é responsável por calcular o novo vetor de estados através da solução da equação de dinâmica fornecida pelo usuário para uma determinada atuação fornecida pelo módulo de controle. A solução desta equação é a solução de uma equação diferencial não linear e pode ser implementada através de um método de integração numérica como, por exemplo, Euler ou Runge Kutta cujo código pode ser encontrado em Numerical Recipes in C [NR in C].

A figura 1-9 mostra, em uma visão macro, o ciclo da simulação.

```
Enquanto simula
     1 – Interpreta os comandos de posicionamento vindos de
um arquivo de entrada ou de um joystick;
     2 – Traduz isto em movimentos desejados do robô;
     3 – Sistema de controle calcula a atuação a ser
aplicada para que o movimento desejado seja realizado;
     4 – Sistema de dinâmica calcula a nova posição e
velocidade do robô de acordo com a atuação fornecida pelo
sistema de controle;
     5 – Sistema de visualização, a partir da nova posição
e velocidade, atualiza a animação;
Fim-enquanto
```

Figura 1-9 - Ciclo da simulação

Os passos 3 e 4 no algoritmo anterior devem ser realizados com a capacidade de interpretação de expressões matemáticas, já que tanto a lei de controle como as equações de dinâmica do robô são fornecidas com entrada através de expressões que variam para cada robô que se deseja simular.

Para a implementação dos passos 3 e 4 o uso de uma linguagem interpretada, ou de compilação dinâmica, simplifica muito a solução do problema, pois caso contrário teríamos que construir um interpretador de equações de controle e de dinâmica ou ligar dinamicamente uma biblioteca que provê esta funcionalidade.

O esquema da figura 1-10 mostra a interação entre os módulos do sistema de simulação

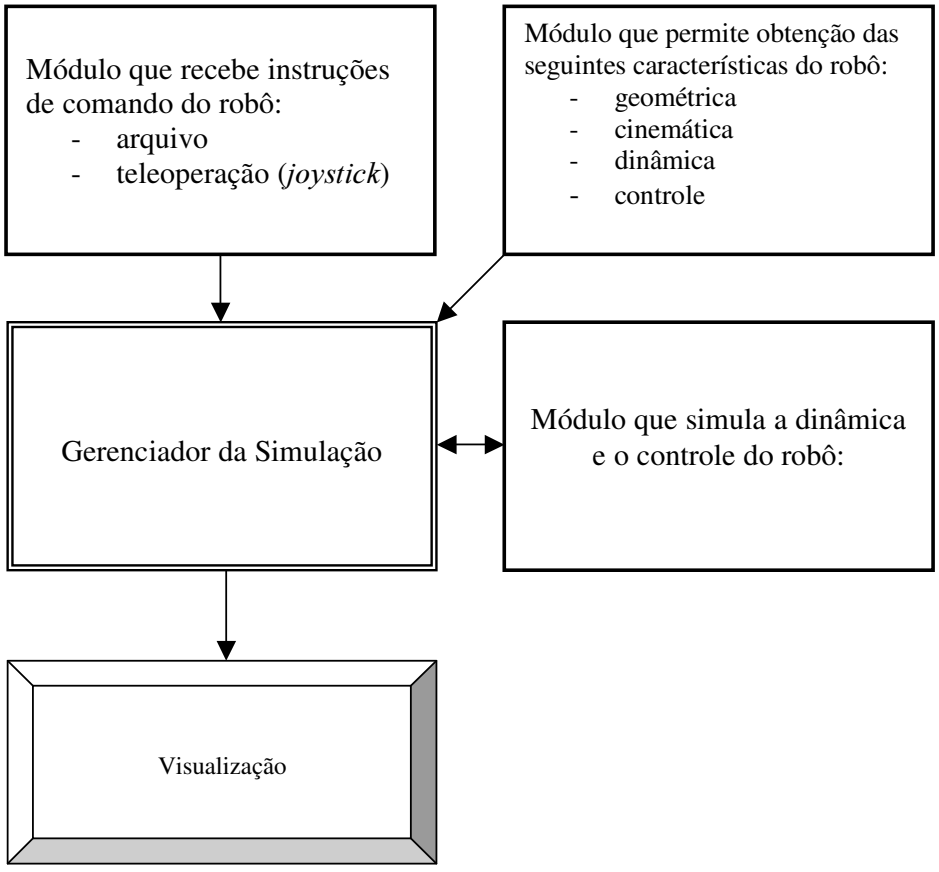


Figura 1-10 - Interação entre os módulos do sistema de simulação

1.5 Trabalhos Relacionados

Apesar da dinâmica estar incorporada às simulações há algum tempo em ferramentas profissionais de alto custo, a animação interativa baseada em física é uma área de pesquisa ainda em crescimento. O conceito de interatividade aqui citado tem haver com as reações que os objetos devem apresentar quando sofrem influência, em seu movimento, de outros objetos que fazem parte de uma cena. Podemos citar como exemplo de aplicações mais comuns os jogos, onde é necessário um cálculo intenso de interação entre os objetos de uma cena. Um carro colidindo com outros objetos estáticos ou não em um jogo de corrida se encaixa muito bem nesta situação. Para adicionar nas simulações tal interatividade se faz necessário um modelo físico mais complexo e genérico, pois aspectos de contato, atrito, colisão, impulso, campos de força, deformação dos corpos, entre outros, devem estar representados neste modelo.

Nos exemplos a seguir são apresentadas características de dinâmica que algumas aplicações suportam.

FreeCad 3D CAD [KOH] é um software acadêmico de simulação dinâmica que permite que o usuário crie e manipule mecanismos ou estruturas representados por sólidos 3D, chamados de partes, conectados por elementos que representam junções, restrições, contatos, atuadores, molas, amortecedores ou forças. Além de informações geométricas e de conexão destes mecanismos ou estruturas, informações sobre massa, momento de inércia, força da gravidade assim como outras forças e torques são modeladas e usadas para gerar a movimentação dos mesmos de acordo com as leis de Newton. Os dados, que podem ser plotados ou gerados em forma tabular, são: deslocamento linear e angular; velocidade linear e angular; aceleração linear, angular e Coriollis; força; torque; momento e energia cinética. As animações que usam estes dados produzem um comportamento dinâmico mais realístico do mecanismo ou estrutura em estudo. Esta ferramenta é muito interessante para realizar análises sobre o comportamento cinemático e dinâmico de um mecanismo ou estrutura, sendo que a simulação pode ser feita com o corpo parado (efeitos estáticos) ou com o corpo em movimento (efeitos dinâmicos). Como resultado da simulação, os dados podem ser apresentados em forma gráfica, tabular, ou em forma de uma animação do corpo em estudo. Uma desvantagem apresentada por este software é a ausência de interatividade, ou seja, o mecanismo simulado não interage com nada a seu redor. Também não existe

possibilidade de criação e edição de um mundo para inserir o mecanismo que está sendo simulado.

Modelica é uma linguagem orientada a objeto para modelagem de sistemas físicos grandes, complexos e heterogêneos [ELMQVIST]. Ela é caracterizada como uma linguagem de modelagem para múltiplos domínios, sendo utilizada para modelagem dos subsistemas mecânico, elétrico, hidráulico e de controle que compõem carros, aviões, e robôs industriais. Em Modelica equações são usadas para modelar fenômenos físicos, onde as variáveis de tais equações não precisam ser manipuladas manualmente para a solução das equações, isto é feito automaticamente. Modelica suporta vários formalismos para edição das equações: equações diferenciais ordinárias ou algébricas, autômato finito, rede Petri, dentre outros. O esforço de modelagem é concentrado na solução do problema, já que existem modelos pré-construídos e testados de objetos que compõem sistemas físicos. Estes modelos contêm o comportamento físico dos objetos.

Dymola é um ambiente de simulação que utiliza a linguagem Modelica anteriormente descrita. Dymola [DYMOLA] integra várias bibliotecas de modelos de diferentes domínios de engenharia. A modelagem se torna rápida através da composição gráfica de modelos de vários domínios (mecânico, elétrico, hidráulico, controle, etc.). Novos modelos de componentes podem ser adicionados. Além da simulação em tempo real também é possível gerar animações off-line.

EcosimPro é uma ferramenta para ambiente Windows com funcionalidade matemática capaz de modelar qualquer tipo de sistemas dinâmico representado por equações diferenciais algébricas ou ordinárias[ECOSIMPRO].

AERO é um software acadêmico para simulação de sistemas de corpos rígidos [KELLER]. Ele possui um editor 3D embutido com o qual pode-se criar uma cena virtual a partir de formas básica tais como: esferas, cubos, cilindros e planos. Estes objetos podem ser conectados com ligações do tipo mola, amortecedor e juntas deslizantes ou girantes. Após a criação dos objetos e da definição das ligações entre eles, pode-se dar início à simulação. A movimentação dos objetos obedece às leis da física, os efeitos de gravidade, resistência do ar, atrito e forças externas adicionadas pelo usuário são todos levados em consideração para a obtenção do movimento correto dos objetos. Este aplicativo permite que seja gerado um vídeo MPEG da animação.

Dynamo é uma biblioteca de classes que executa os cálculos dos movimentos de objetos geométricos que estão sobre a influência de forças, torques e impulsos [BARENBURG]. O cálculo de dinâmica é feito de forma implícita. Além disso, a biblioteca também computa forças através da especificação de restrições *(constraints)*. Estas restrições permitem conectar facilmente os objetos geométricos de várias formas. Uma restrição só precisa ser especificada uma única vez, e a biblioteca se encarrega de continuamente satisfazer a condição imposta pela restrição através da aplicação de forças de reação. As principais características desta biblioteca segundo [BARENBURG] são:

- Suporte total aos cálculos de dinâmica direta, relativos à aplicação de forças, torques e impulsos.
- Boa performance para o cálculo e retorno, para a aplicação cliente, das novas posições e orientações dos objetos desta aplicação.
- Cálculo mais rápido da dinâmica inversa através de *constrainst*.
- Implementação de vários tipos de restrições de ligação e colisão entre corpos.
- Suporte para controladores, atuadores e sensores.
- Projeto e implementação orientado a objeto, que permite fácil extensão da biblioteca.

1.6 Estrutura da dissertação

Neste capítulo foi estabelecido o escopo do problema como sendo uma simulação da movimentação de um robô conhecidos seu modelo geométrico, seu modelo dinâmico e uma lei de controle para atuar em sua movimentação. O capítulo 2 descreve diversas ferramentas de desenvolvimento e o que levamos em consideração na escolha de algumas. O capítulo 3 faz uma descrição dos princípios de dinâmica e de controle necessários a modelagem do robô a ser simulado e apresenta as características dos módulos responsáveis pelo cálculo do controle e da dinâmica do robô. O capítulo 4 mostra como é realizada a modelagem geométrica do robô usando um grafo de cena em Open Inventor com algumas extensões. O capítulo 5 descreve detalhes de implementação da simulação. O capítulo 6 apresenta os resultados obtidos na simulação de dois modelos mecânicos (o pêndulo invertido e o braço articulado com dois graus de liberdade) e as conclusões a respeito do trabalho. Os apêndices trazem informações tais como: o diagrama de classes e os ícones do grafo de cena da biblioteca Open Inventor, e os arquivos de entrada, intermediários e de saída para a simulação do pêndulo invertido de do braço articulado com dois graus de liberdade.

2 Descrição do sistema

O objetivo neste capítulo é detalhar a proposta de solução apresentada no capítulo 1, discutir e comparar algumas ferramentas de desenvolvimento voltadas à computação gráfica e explicar os motivos da escolha de algumas delas para a realização deste trabalho.

Iniciaremos, na sessão 2.1, pelo problema as ser resolvido; na seqüência faremos, na sessão 2.2, a discussão e comparação das ferramentas de desenvolvimento; e por último a escolha das ferramentas e seus motivos.

2.1 Abordagem do problema

Primeiramente vamos apresentar a estrutura geral de uma simulação baseada em física, posteriormente apresentamos a estrutura do sistema de simulação de robôs que nos propomos a construir bem como as diferenças e semelhanças entre eles.

2.1.1 Estrutura geral de uma simulação baseada em física

Uma análise inicial nos leva a identificar duas etapas básicas em uma simulação. A primeira etapa consiste em inicializar todos os objetos com os valores que representem suas posições iniciais. A segunda etapa consiste de um ciclo que deverá ser executado enquanto durar a simulação, composto, por sua vez, de três fases. Na primeira fase do ciclo devemos verificar se houve ou não colisão entre os objetos e responder adequadamente a estas colisões. Na segunda fase do ciclo calculamos o novo estado dinâmico do sistema. Na terceira fase do ciclo devemos atualizar a posição dos objetos e movê-los para esta posição. Este algoritmo geral pode ser melhor visualizado através do diagrama de blocos da figura 2-1.

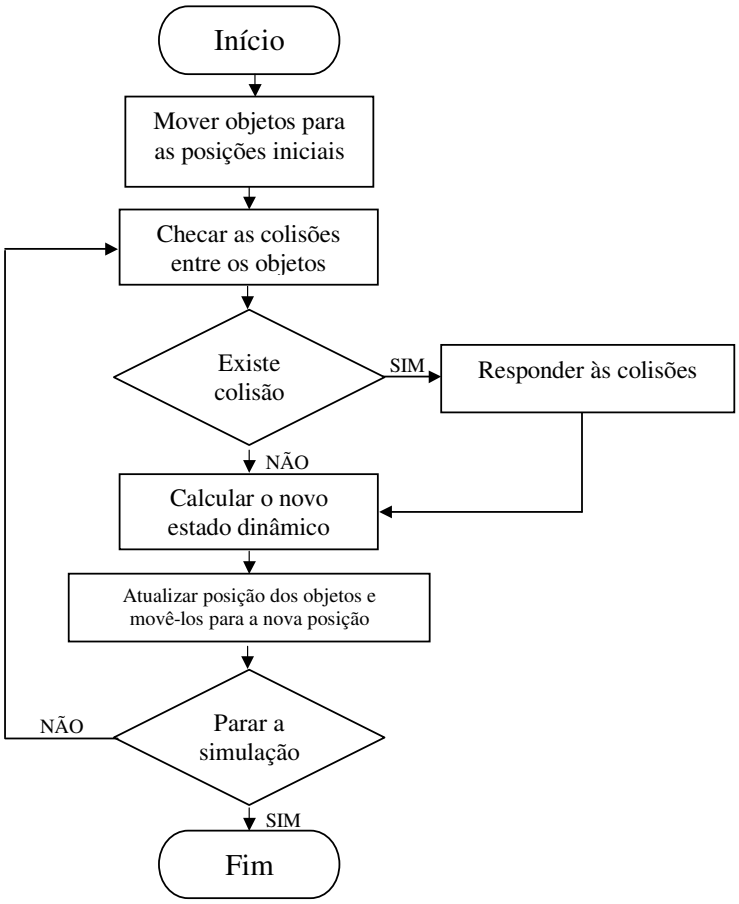


Figura 2-1 Diagrama de blocos de uma simulação baseada em física

A simulação começa em um tempo t_0 e executa os cinco grandes passos da figura 2-1 que são detalhados a seguir:

Mover os objetos para as posições iniciais: os objetos são movidos de acordo com as condições iniciais de posição, velocidade, aceleração linear e angular, e torques/forças externas aplicadas aos mesmos, ignorando-se qualquer colisão que possa ocorrer durante esta movimentação. A determinação destas condições iniciais também pode ser chamada de determinação do estado dinâmico inicial dos objetos, que servirá para calcular o próximo estado dinâmico dos objetos simulados. Este cálculo é feito através da integração numérica das equações diferenciais ordinárias que descrevem o movimento para cada objeto simulado. Partindo-se da posição, velocidade e aceleração, que são conhecidas, a força/torque pode ser calculada de acordo com a interação entre os objetos (abordagem implícita ou *on-line*) ou através da equação explícita de dinâmica

dos corpos (abordagem explícita ou *off-line*). Este passo nada mais é do que o cálculo do primeiro estado dinâmico a partir das condições iniciais.

Checar as colisões entre os objetos: neste passo a primeira tarefa a ser realizada é determinar quais são os objetos que colidem, através do cálculo da verificação de interseção geométrica entre as estruturas que representam os objetos. Este cálculo é computacionalmente muito custoso e requer estruturas apropriadas para a representação geométrica dos objetos, pois para cada objeto que compõe o ambiente simulado deve-se verificar se ele pode colidir com todos os outros objetos. Efetuado o cálculo cria-se então uma lista dos objetos envolvidos em colisões que será utilizada pelo próximo passo que é responder às colisões.

Responder às colisões: a resposta à colisão pode alterar completamente os movimentos dos objetos. Após uma colisão, a direção, o sentido e a intensidade das forças que atuam em um objeto podem ser alterados. Um objeto que estava em um estado estático pode passar a apresentar um estado dinâmico e vice-versa. Neste passo são realizados alguns cálculos de dinâmica que irão afetar o próximo passo que é o cálculo do novo estado dos objetos. Em algumas situações, pode haver a necessidade da execução de um ciclo mais interno composto somente dos passos: checar colisões, responder às colisões e calcular o novo estado dinâmico, para só então passar para o passo atualizar as posições dos objetos. Estas situações estão caracterizadas por muitas colisões simultâneas entre os mesmos objetos ou situações de contato entre os corpos onde a velocidade relativa entre eles possui uma intensidade muito pequena.

Calcular o novo estado dinâmico: o novo estado dinâmico dos objetos é calculado baseado no estado dinâmico anterior. Este cálculo é feito através de um método de integração numérica [NR in C]. Por exemplo, considerando $Y(t)$ a função que descreve o estado dinâmico de um objeto em função do tempo (t), usando o método de Euler, temos:

$$Y(t) = Y_0(t) + \dot{Y}(t)$$

Onde $Y(t)$ é o novo estado do sistema, $Y_0(t)$ é o estado anterior e $\dot{Y}(t)$ a primeira derivada de $Y(t)$.

Segundo [COUTINHO], para uma simulação de corpos rígidos, o estado dinâmico de um corpo pode ser descrito por:

$$\vec{y}(t) = \begin{pmatrix} \vec{r}(t) \\ R(t) \\ \vec{P}(t) \\ \vec{L}(t) \end{pmatrix}$$

Onde $\vec{r}(t)$ e $R(t)$ são a posição e a orientação do centro de massa do corpo, e $\vec{P}(t)$ e $\vec{L}(t)$ são os momentos linear e angular do corpo respectivamente. No capítulo 3 é apresentado um detalhamento sobre como obter as equações de dinâmica de um sistema.

No caso de uma simulação de vários corpos rígidos, ou de um corpo rígido articulado composto de várias partes, $Y(t)$ representa o estado dinâmico do sistema como um todo, ou seja, representa o estado dinâmico dos *N* objetos que estão sob os efeitos da dinâmica. Neste caso $Y(t)$ também é conhecido como vetor de estado dinâmico do sistema.

$$Y(t) = \left(\vec{r}_1(t), R_1(t), \vec{P}_1(t), \vec{L}_1(t), \ldots, \vec{r}_N(t), R_N(t), \vec{P}_N(t), \vec{L}_N(t)\right)$$

Onde o índice *N* representa o N-ésimo objeto do sistema.

No primeiro passo da simulação obtém-se as condições iniciais, ou seja, o estado dinâmico inicial $Y_0(t)$. Para se calcular o próximo estado, através da integração numérica, é preciso conhecer $\dot{y}(t)$, ou seja, a derivada no tempo de $\vec{y}(t)$:

$$\dot{y}(t) = \frac{d}{dt}\vec{y}(t) = \begin{pmatrix} \frac{d}{dt}\vec{r}(t) \\ \frac{d}{dt}R(t) \\ \frac{d}{dt}\vec{P}(t) \\ \frac{d}{dt}\vec{L}(t) \end{pmatrix} = \begin{pmatrix} \vec{v}(t) \\ \tilde{\omega}(t)R(t) \\ \vec{F}(t) \\ \vec{\tau}(t) \end{pmatrix}$$

Onde $\vec{v}(t)$ é a velocidade linear, $\tilde{\omega}(t)R(t)$ é a velocidade angular, $\vec{F}(t)$ é a força e $\vec{\tau}(t)$ é o torque.

Para se obter a derivada do estado dinâmico, é preciso calcular $\vec{F}(t)$ e $\vec{\tau}(t)$ total que agem sobre o corpo. O cálculo da força/torque total que age sobre o corpo pode ser feito de duas formas: explícita ou implicitamente. No caso explícito é necessário possuir as equações de dinâmica do sistema e no caso implícito é necessário conhecer as relações de conexão e restrição de movimento entre os objetos simulados.

Do novo estado dinâmico extraí-se informações sobre posição e orientação dos objetos para que estas informações sejam comunicadas ao sistema de animação.

Atualizar a posição dos objetos e movê-los para a nova posição: neste passo extrai-se do novo estado dinâmico dos objetos informações sobre posição e orientação dos objetos para que possam ser comunicadas ao sistema de animação (plotagem, renderização). Este, por sua vez, se encarregua de redesenhar a cena com os objetos em suas novas posições.

2.1.2 Estrutura proposta para a simulação de robôs

O diagrama de blocos da figura 2-2 apresenta o algoritmo da simulação de robôs.

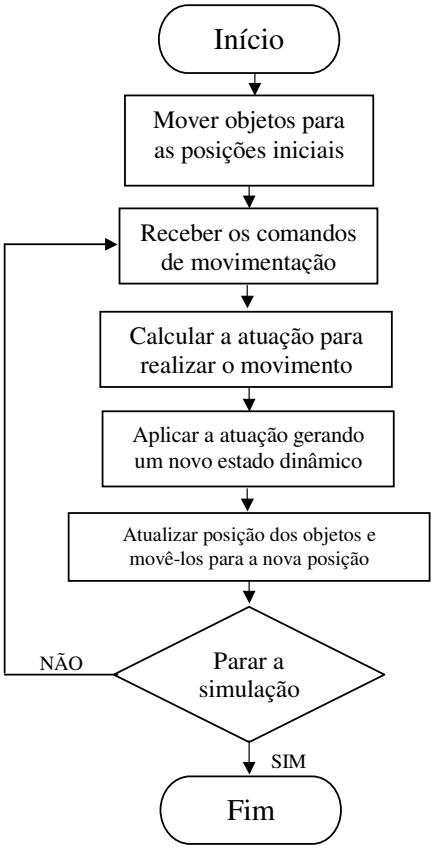


Figura 2-2 Diagrama de blocos da simulação de robôs

Em nossa proposta de solução, não existe a aplicação de uma força/torque diretamente sobre os objetos, a entrada para o sistema é um comando de movimentação, que será traduzido para uma atuação (força/toque) pelo sistema de controle. Daí por

diante, a simulação de robôs toma uma forma parecida com a estrutura geral apresentada anteriormente (ver figura 2-1), ou seja, enquanto não houver outro comando de movimentação mantém-se o ciclo do cálculo da atuação e aplicação desta atuação no modelo de dinâmica.

A diferença entre uma simulação baseada em física (figura 2-1) e a simulação de robôs proposta (figura 2-2) é marcada pela presença do módulo de comando e do módulo de controle, que juntos fazem o papel da atuação que o módulo de dinâmica precisa receber como entrada. A sua principal contribuição do módulo de comandos para o sistema de simulação é proporcionar interatividade entre o usuário e o sistema de simulação. A interatividade é uma característica relevante em sistemas de simulação e podemos realizá-la de três formas: através de comandos, através da interação entre vários objetos na mesma simulação e através de o que há de mais novo nesta área de pesquisa, que é a interatividade proporcionada pela realidade virtual. No primeiro caso o usuário pode interagir com a simulação através de comandos emitidos ao robô por meio de uma interface de software (um arquivo, um item de *menu*, um botão de função, etc.). No segundo caso, a interação se dá entre os vários objetos que fazem parte de um ambiente simulado, exemplo: colisão, contato, interpenetração, trabalho cooperativo, etc. No terceiro caso, o usuário possui uma interface mais elaborada para comandar a simulação, ao invés de passar comando através de itens de *menu* ou de botões ou teclas de funções, ele o faz através de, por exemplo, luvas, capacetes, e óculos de realidade virtual, o que lhe dará a falsa impressão, por isso realidade virtual, de que está fisicamente manipulando o robô.

Na simulação baseada em física, os corpos são expostos à aplicação de uma força/torque e o seu movimento é decorrente da resultante desta força/torque e das forças/torques de ação e reação entre os corpos que colidem entre si ou com outros objetos que fazem parte da simulação. Por exemplo, se a simulação é um jogo de futebol, a força que é aplicada na bola é entrada para simular seu movimento. Esta entrada ou atuação é recebida através de um comando do usuário do jogo para "*chutar a bola*". Este comando pode, por exemplo, ser passado em valores de força, "*chute a bola com uma força de intensidade F aplicada em uma direção D*".

Diferentemente do que é feito na simulação baseada em física, no sistema de simulação de robôs proposto, o comando é primeiramente enviado ao sistema de controle, e este por sua vez é quem faz o cálculo da atuação, ou seja, da força/torque

necessários à realização do comando. O que queremos enfatizar é que no sistema proposto a simulação, além de levar em consideração a dinâmica, também considera o sistema de controle necessário ao controle de movimento do robô. A figura 2-3 destaca esta diferença.

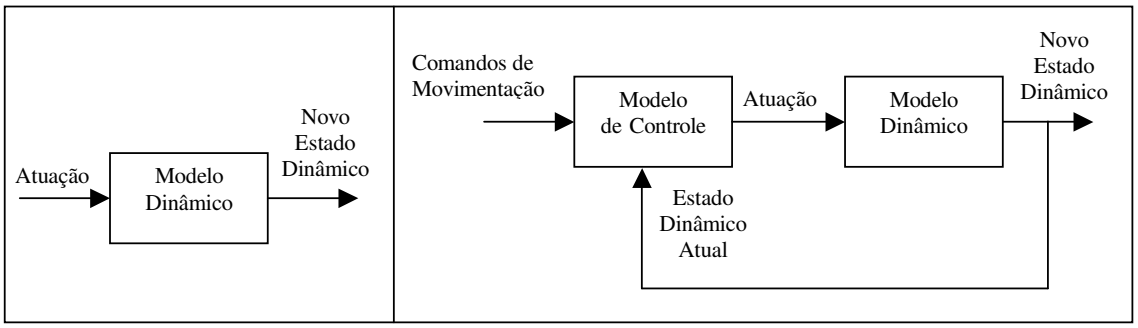


Figura 2-3 - (a) Simulação Baseada em Física. (b) Simulação de robôs proposta

Como pode ser visto na figura 2-3 (a), o novo estado dinâmico é realimentado no modelo de controle e passa a ser o estado dinâmico atual para servir de comparação com o estado dinâmico desejado fornecido pelos comandos de movimentação.

2.1.3 Abordagem explícita (*off-line*) versus abordagem implícita (*on-line*) na determinação do estado dinâmico dos objetos

Na sessão 2.1.1 vimos que, nas simulações computacionais baseadas em física, o cálculo do estado dinâmico do corpo pode ser feito de duas formas: a forma implícita ou *on-line* e a forma explícita ou *off-line*.

Como estamos tratando de uma simulação de robôs, a eficiência da simulação é uma qualidade desejada como resposta, por isso escolher a forma explícita para os cálculos de dinâmica seria o mais indicado. Contudo esta abordagem possui limitações quando se trata de sistemas mecânicos complexos, pois a determinação das equações de dinâmica para tais sistemas é uma tarefa muito árdua. Em muitos casos, onde o número de equações encontradas é muito grande ou o grau de complexidade destas equações é muito alto, algumas simplificações ou linearizações podem ser feitas na tentativa de se criar um modelo dinâmico mais simples, porém menos fiel à realidade.

A simulação na forma implícita não necessita do passo inicial de determinação das equações de dinâmica do robô, porém para que apresente um resultado com qualidade em precisão, várias restrições de movimento devem ser criadas e devem ser testadas a cada cálculo do novo estado dinâmico. Todas as relações de interconexão e contato entre as partes de um mesmo corpo e entre um corpo e outro no ambiente criado devem ser descritas. Nesta abordagem existem vários tipos básicos de interconexão e contato e várias combinações destes tipos básicos. Isto torna o cálculo do estado dinâmico uma tarefa extremamente lenta e potencialmente imprecisa.

A simulação na forma implícita pode ser largamente utilizada para criarmos animações para filmes e jogos, pois estas animações não necessitam de muita precisão e eficiência, podendo ser apresentadas em uma taxa de quadros muito maior do que aquelas com as quais foram geradas. Além disso, dispensam a necessidade de um usuário especialista para determinar as equações de dinâmica. A simulação na forma explícita se adequa bem ao sistema de simulação de robôs proposto por causa da necessidade de eficiência e de precisão. A obrigatoriedade do fornecimento das equações de dinâmica, que pode ser apontada como uma desvantagem, é na verdade uma etapa que provavelmente deve ser cumprida pelo usuário deste tipo de simulação, pois ele precisa conhecer as equações de dinâmica para projetar a lei de controle.

No capítulo 3 são apresentadas algumas formulações para a determinação da equação de dinâmica de sistemas de corpos rígidos.

2.2 Análise das ferramentas gráfica 3D existentes

Como o objetivo neste trabalho é realizar uma simulação computacional de robôs com os resultados sendo apresentados através de uma animação em 3D do modelo geométrico do robô, procurou-se ferramentas de desenvolvimento que atendessem a certos requisitos, tais como: nível de abstração, portabilidade e acesso.

As justificativas para a busca destes requisitos são:

- O nível de abstração pode ter um impacto na implementação da solução, se o nível de abstração da linguagem com que se trabalha for muito baixo, o desenvolvedor pode gastar muito do seu esforço em detalhes e não conseguir projetar uma boa solução.

- A portabilidade também é um requisito desejado, pois amarrar uma solução a uma plataforma de desenvolvimento pode inviabilizar a continuidade do projeto no futuro.
- Entre o acesso e o nível de abstração é necessário estabelecermos uma relação de compromisso, geralmente quanto maior o nível de abstração menor é o acesso as estruturas internas que representam uma solução e vice-versa. No caso da simulação computacional dos robôs, é desejado um certo acesso às estruturas que representam o modelo geométrico do robô, pois com os resultados da simulação deseja-se fazer uma animação do modelo geométrico.

Quando iniciamos os estudos das ferramentas de desenvolvimento de um ambiente gráfico para simulação dinâmica de robôs as primeiras que vieram à mente foram as interfaces de programação OpenGL e Direct3D. Estas interfaces provêm várias funcionalidades necessárias ao desenvolvimento de aplicações gráficas, porém em um nível de abstração muito baixo. A manipulação de objetos gráficos se dá ao nível de pontos e conjuntos de vértices. Para desenhar um robô usando uma destas interfaces, teríamos que fazê-lo pela definição de um conjunto de vértices para cada parte que compõe o robô. A manipulação deste robô teria que ser feita vértice a vértice, o que nos levaria a escrever toda uma camada de abstração que permitisse a manipulação de objetos mais complexos antes de nos preocuparmos com nosso problema principal.

Existem API´s ("*Aplication Program Interface*") com maior nível de abstração, tais como VRML, Java 3D e Open Inventor da Silicon Graphics, que apresentam classes de objetos geométricos básicos tais como cubo, esfera, prisma, planos e composições destes, permitindo então que foquemos nosso esforço no problema principal.

Iniciaremos, a seguir, uma análise de prós e contras de cada uma das API's estudadas.

2.2.1 OpenGL

OpenGL (Silicon Graphics) é uma API que já alcançou um certo nível de estabilidade, pois já vem sendo utilizada por usuários em estações gráficas avançadas e supercomputadores desde 1992 [SILICON GRAPHICS]. Apesar de sua popularidade com soluções, para Windows, Mac e Linux, o nível de abstração para construção de algumas aplicações gráficas ainda é baixo. OpenGL é uma interface mais procedural que descritiva [SHREINER], pois ao invés do programador descrever a cena como ela

deveria ser apresentada, ele descreve quais são os passos necessários para se obter determinado efeito ou aparência. O programador fica preso a detalhes de implementação, e é difícil focar o problema a ser resolvido, ou seja, a seqüência em que as ações acontecem na cena. Para criar uma cena são feitas chamadas à interface que possui aproximadamente 120 funções. Em uma cena, tanto a criação quanto a manipulação de objetos são feitas, basicamente, no nível de polígonos. OpenGL possui funções para iluminação, sombreamento, textura, animação, e outros efeitos. O baixo nível de abstração seria compensado pela alta portabilidade, porém esta não é plenamente atingida por causa da dependência que as funções de renderização possuem do sistema de gerenciamento de janelas do sistema operacional e do acelerador gráfico (placa de vídeo) utilizado.

2.2.2 Direct3D

Direct3D, assim como OpenGL, é também uma API que apresenta os mesmos problemas de baixo nível de abstração com a desvantagem em relação a OpenGL de não apresentar portabilidade. É uma API desenvolvida pela Microsoft para funcionar em sistemas operacionais da Microsoft.

2.2.3 VRML

VRML é um acrônimo para "*Virtual Reality Modeling Language*". Trata-se de um padrão internacional ISO/IEC 14772 [ISO/IEC 14772] de formato de arquivo para descrever uma interatividade multimídia em 3D na Internet. A especificação de sua primeira versão, VRML 1.0, foi criada pela Silicon Graphics, Inc. baseada no formato de arquivo do Open Inventor [CAREY].

VRML não é uma linguagem de propósito geral como C++, nem uma linguagem de script como JavaScript, nem uma linguagem de especificação ou formatação de texto como HTML e nem é uma API. Ela é uma linguagem de descrição de cena na qual é possível descrever geometria e o comportamento de cenas em 3D [VRML]. Sempre é necessária a presença de um navegador para a interpretação de um arquivo VRML, o qual apresenta as formas e sons em uma cena gráfica. Esta apresentação é conhecida com mundo virtual que é navegável através de um navegador por uma pessoa ou entidade mecânica, chamada de usuário.

O mundo é apresentado de um ponto de vista, a posição e orientação deste ponto de vista é chamado de observador. O navegador disponibiliza alguns paradigmas de navegação, tais como andar e voar, que possibilita que o usuário movimente o observador através do mundo virtual.

O navegador provê, além da navegação, um mecanismo que permite o usuário interagir com o mundo através de sensores na cena. Estes sensores respondem às interações do usuário com objetos geométricos no mundo (sensor de toque), aos movimentos do usuário através do mundo (sensor de proximidade) e à passagem do tempo (sensor de tempo) [CAREY]. A figura 2-4 ilustra o modelo conceitual de um navegador VRML. O navegador é enquadrado como uma aplicação de apresentação que aceita entradas do usuário em forma de seleção de arquivo, manipulação e navegação usando dispositivos de entrada. Os três componentes principais do navegador são: *Parser*, Grafo da Cena, e Apresentação Áudio/Visual.

O *Parser* lê o arquivo VRML e cria o Grafo da Cena. O Grafo da Cena consiste de uma Hierarquia de Transformação (nós) e uma Rota Gráfica, também inclui uma Máquina de Execução (*engine*) que processa os eventos, lê e edita a Rota Gráfica e faz mudanças nos nós. As entradas do usuário afetam os sensores e a navegação, por isso estão amarradas à Rota Gráfica (sensores) e a Apresentação Áudio/Visual (navegação). A Apresentação Áudio/Visual executa a renderização dos gráficos e do áudio dos nós e então o resultado da renderização é retornado ao usuário. Uma linguagem de *script* é utilizada como apoio à programação de objetos que possuem comportamento.

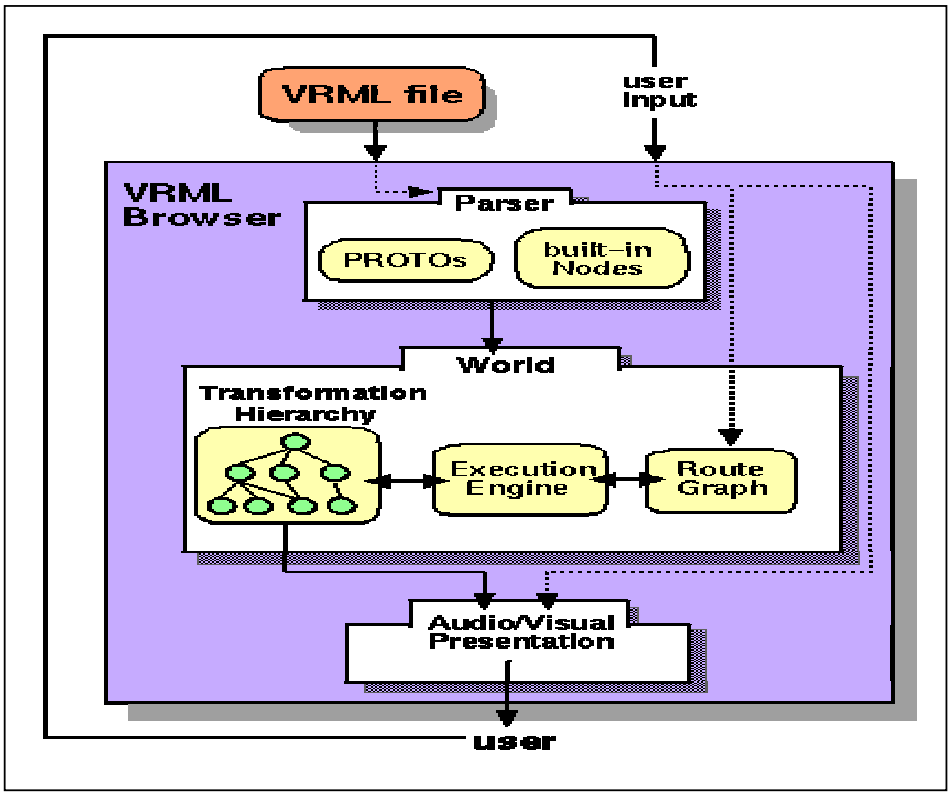


Figura 2-4 - Modelo conceitual de um navegador VRML [CAREY]

2.2.4 Java 3D

A JavaSoft e uma cooperativa de empresas estenderam a linguagem Java com uma família de API's, chamada Java Media, para resolver o problema de ausência de integração de tecnologias de multimídia tais como áudio, vídeo, e gráficos 3D em aplicações baseadas em Web [SUN]. No caso do Java 3D as empresas participantes foram a Silicon Graphics, a Intel, a Apple Computer e a Sun Microsystems. A falta de integração de várias mídias em uma única aplicação apontou a criação dos *plug-in's* como uma solução paliativa, porém pobre do ponto de vista da aplicação cliente, uma vez que são a aplicação e o *plug-in* são aplicações separadas. A API do Java 3D faz parte do Java Media, e é usada para escrever aplicações gráficas tridimensionais *stand-alone* ou *applets* 3D.

Com Java 3D os desenvolvedores podem descrever grandes mundos virtuais que são renderizados também em Java 3D. Esta API tem um nível de abstração maior que OpenGL e Direct3D, pois permite que o programador manipule objetos geométricos ao

invés de polígonos. Java 3D permite um alto grau de interatividade e preserva a independência entre plataformas "*write once, run anywhere*". Apesar de poder ser usada em navegador da Internet, sistemas de realidade virtual, jogos, e sistemas CAD, ainda não possui ambientes de autoria e nem um formato de arquivo 3D definido. Além disso, todo o desempenho anunciado pelos desenvolvedores do produto, só pode ser levado em consideração quando comparado com o de outras aplicações também escritas em Java, uma vez que a linguagem por concepção não apresenta um bom desempenho.

O modelo de programação da cena em Java 3D é baseado em um grafo acíclico. O grafo da cena contém uma descrição completa da cena ou mundo virtual, incluindo dados geométricos, valores de atributos e informações de visualização necessárias a renderização da cena a partir de um ponto de vista. Também, da mesma forma que VRML, Java 3D se propõe a ser um descritor de cenas gráficas. A figura 2-5 mostra um exemplo de um grafo de cena em Java 3D.

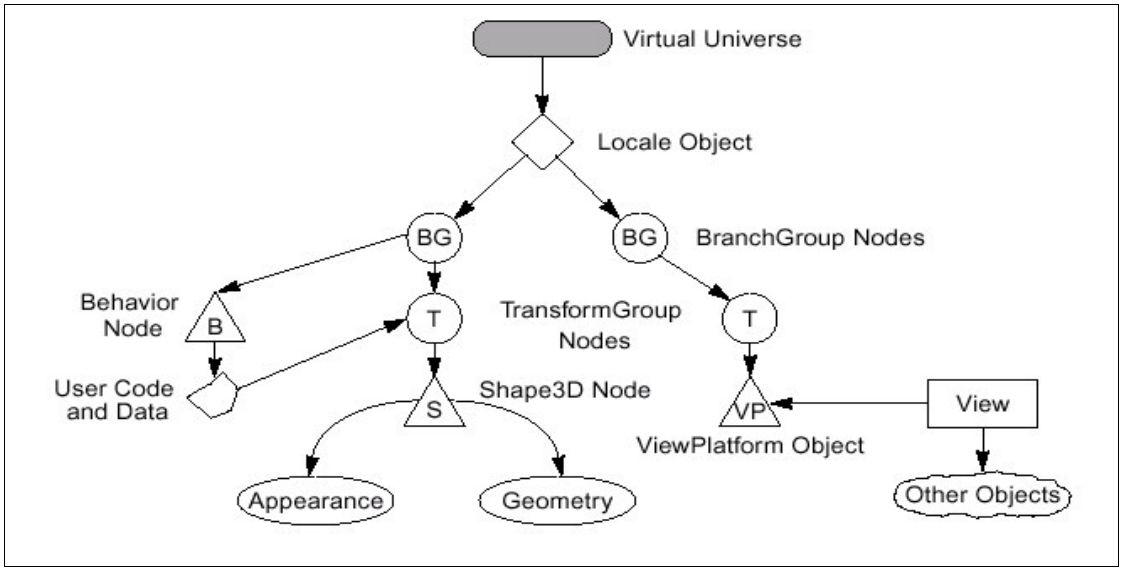


Figura 2-5 - Uma cena em Java 3D é representada por um grafo acíclico [SUN]

2.2.5 Open Inventor

Open Inventor é uma biblioteca baseada em OpenGL que provê classes de objetos que podem ser utilizados, modificados e estendidos para se adequar à aplicação a ser desenvolvida [OPEN INVENTOR]. Os objetos disponíveis nesta biblioteca incluem objetos primitivos tais como cadeia de caracter (*string*), números reais (*float*), coordenadas 3D, vetores, matrizes e também objetos mais complexos tais como formas geométricas, luz, câmeras, grupos de objetos, conectores entre objetos para envio de mensagens, manipuladores interativos (*mouse* e *trackball*) e alguns componentes tais como editor de luz, editor de material e um visualizador de cenas. A cena pode ser criada em um arquivo com formato próprio e com extensão **iv** cujo conteúdo pode estar no formato binário ou ASCII.

O foco desta biblioteca é criar e manipular objetos 3D e não desenhos ou conjuntos de pontos. Todas as informações dos objetos tais como: forma, tamanho, cor, textura e localização no espaço 3D são armazenadas na árvore que compõe a cena. A estrutura desta árvore determina a ordem das operações de renderização e, por conseguinte a aparência destes objetos.

As cenas em Open Inventor podem ser geradas basicamente de duas formas, sendo que em ambas, a biblioteca do Open Inventor deverá ser ligada (*link*) com a aplicação em C ou C++ que está sendo desenvolvida. A primeira e talvez a mais comum é criar, manipular e apresentar objetos gráficos em uma janela de renderização através da codificação de um programa em linguagem C ou C++ que utilize funções específicas da biblioteca Open Inventor para executar tais tarefas. A outra forma de gerar uma cena é usar um arquivo ASCII que contém a descrição da cena, desta forma a aplicação comanda a carga dos objetos a partir do arquivo, e qualquer alteração nestes objetos pode ser salva de volta no arquivo garantindo assim a persistência entre execuções distintas.

Podemos ver, na figura 2-6 a seguir, o conteúdo de um arquivo de descrição do grafo de cena. Este arquivo possui a extensão iv e pode ter seu conteúdo no formato ASCII ou binário. A cena descrita pelo arquivo da figura 2-6 contém uma esfera vermelha e um cubo azul.

```
#Inventor V2.0 ascii

Separator {
    Separator {
        DrawStyle {
            style       LINES
        }
        Material {
            diffuseColor        1 0 0
        }
        Sphere {
            radius      1
        }
    }
    Translation {
        translation     2 0 0
    }
    Separator {
        Material {
            diffuseColor        0 0 1
        }
        Cone {
            bottomRadius        1
            height      2
        }
    }
}
```

Figura 2-6 - Conteúdo de um arquivo ASCII em formato Open Inventor

A hierarquia das propriedades no grafo da cena é importante e facilita a construção da mesma por permitir que propriedades globais sejam descritas no topo e mais à esquerda na árvore e as propriedades locais sejam descritas a baixo e mais à direita.

A figura 2-7 mostra um grafo que representa o conteúdo do arquivo iv mostrado na figura 2-6. O formato do arquivo iv e o grafo de cena apresentados, respectivamente na figura 2-6 e na figura 2-7, estão detalhados no capítulo 4.

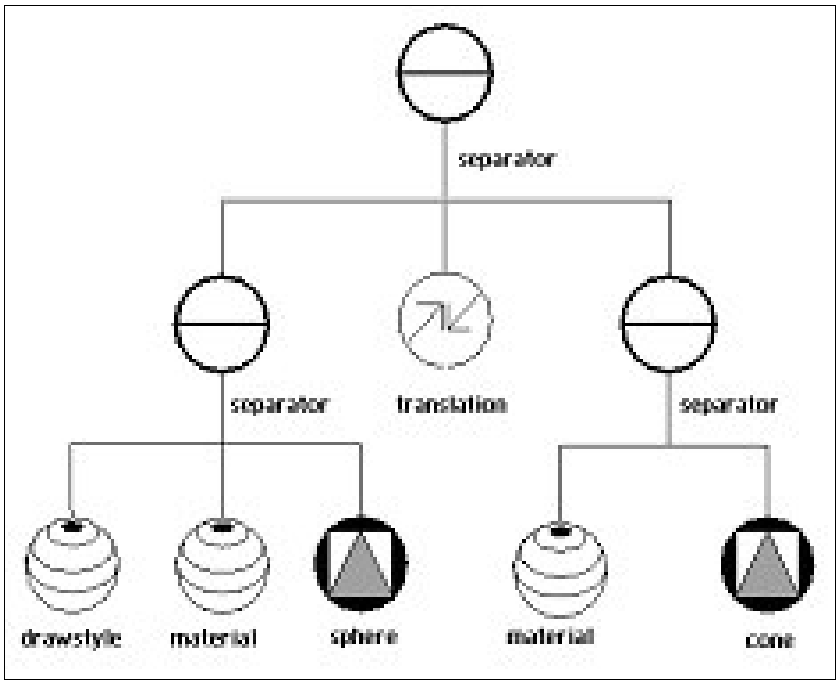


Figura 2.7 Grafo de uma cena em Open Inventor

Cada nó do grafo é representado por um ícone que é associado a uma classe ancestral de objetos do Open Inventor. Estas classes ancestrais representam objetos que possuem propriedades comuns entre si. Na figura 2.7 tanto a esfera quanto o cone são representados pelo mesmo ícone apesar de serem objetos geométricos distintos possuem propriedades comuns tais como cor, tamanho, posição, etc, e portanto são objetos da mesma classe ancestral. Nos apêndices A e B podem ser encontrados os ícones utilizados na representação dos grafos de cenas e a hierarquia das classes de objetos da biblioteca Open Inventor.

A figura 2-8 a seguir mostra uma cena renderizada pela aplicação de visualização de cenas chamada SceneViewer. Esta aplicação e alguns outros utilitários são disponibilizados pela SGI juntamente com a biblioteca Open Inventor. A cena renderizada na figura 2-8 é relativa ao conteúdo do arquivo apresentado na figura 2-6.

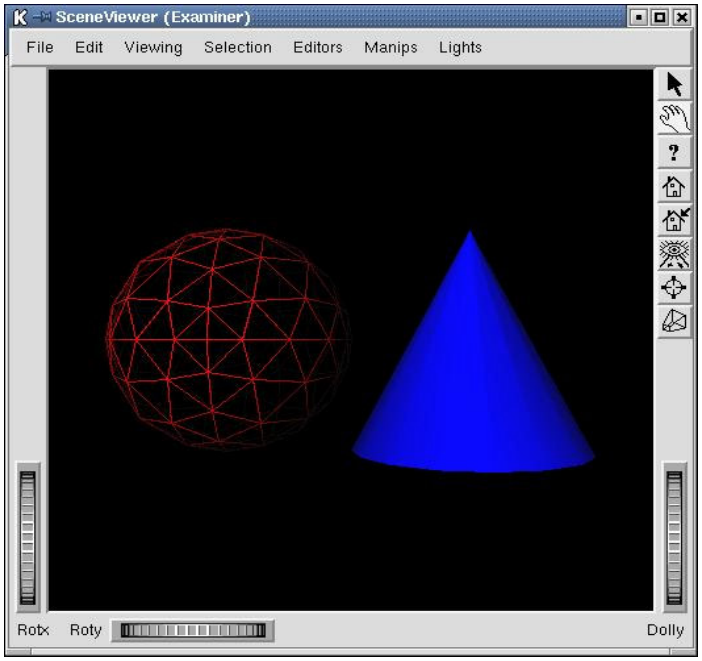


Figura 2-8 - Cena renderizada com a aplicação gview da SGI

2.2.6 Comparação entre as ferramentas

Apesar da ampla funcionalidade que as API´s descritas em 2.2.1 e 2.2.2 provêem, seria mais interessante gerar uma animação a partir de uma API de mais alto nível devido à baixa curva de aprendizagem e conseqüentemente menor tempo de desenvolvimento.

Para ficar mais fácil de se extrair as vantagens e desvantagens das API's e linguagens relacionadas anteriormente, pode-se classificá-las segundo a taxonomia apresentada na figura 2-9 a seguir.

		CARACTERÍSTICA				
		Nível de Abstração	Acesso	Portabilidade	Existência de um formato de arquivo	Existência de Editor 3D
LINGUAGEM / API	Direct3D	Baixo	Alto	Não	Não	Sim
	OpenGl	Baixo	Alto	Sim	Não	Sim
	Java 3D	Alto	Alto	Sim	Não	Não
	VRML	Alto	Alto	Sim	Sim	Sim
	Open Inventor	Alto	Alto	Sim	Sim	Não

Figura 2-9 - Tabela comparativa entre as ferramentas gráfica 3D de desenvolvimento

Interpretando o que significa a tabela apresentada na figura 2-9 temos:

- **Nível de abstração** - tem haver com o quanto mais próximo ou distante da realidade estão as estruturas de dados propostas pela linguagem ou API. Quanto mais alto o nível de abstração, mais as estruturas de dados se assemelham à realidade que pretendem modelar.

  OpenGL e Direct3D possuem um baixo nível de abstração quando comparadas com VRML, Java 3D e Open Inventor, pois se preocupam apenas com a imagem ou o desenho a ser gerado. O objetivo final é uma imagem renderizada, não existem facilidades para manipular os objetos distintos que compõem a imagem. Desta forma o programador fica responsável por criar suas próprias estruturas de dados para poder manipular individualmente os objetos que compõem a imagem.

  VRML, Java 3D e Open Inventor possuem um nível de abstração bem alto, pois as imagens ou desenhos gerados possuem classes de objetos que os representam, por isso eles podem ser facilmente manipulados, alterados, armazenados, recuperados, impressos ou redesenhados sempre que for necessário.

- **Acesso** - Todas dão acesso aos vários níveis de representação das estruturas de dados através de uma hierarquia de classes.

- **Portabilidade** - em princípio podemos classificar OpenGL, Java 3D e Open Inventor como portáveis. OpenGL e Open Inventor são API's escritas em C++. No caso de VRML, desde que exista um navegador para a plataforma de interesse, podemos considerá-la portável, já que o mesmo arquivo VRML pode ser apresentado por vários navegadores em plataformas distintas. Direct3D não é de forma alguma portável já que é um produto desenvolvido para o ambiente Microsoft Windows.

- **Existência de um formato de arquivo** - esta característica é interessante, pois quando é necessário intercambiar informação com outros aplicativos, a existência de um formato de arquivo torna esta tarefa realizável de forma padronizada. A ausência de um formato de arquivo deixa por conta do desenvolvedor a tarefa de criar seu próprio padrão para dar persistência aos objetos gráficos. Com cada desenvolvedor criando seu próprio padrão não é possível a troca de arquivos entre eles. Direct3D, OpenGL e Java 3D não possuem um formato de arquivo, já Open Inventor e VRML possuem. Java 3D perde muito com a inexistência de um formato de arquivo padrão.

- **Existência de editor 3D** - Não é nosso interesse listar todos os editores 3D existentes, mesmo porque nos atemos somente a pesquisar os de domínio público. OpenGL e Direct3D possuem uma grande quantidade de editores 3D de domínio privado que apresentam inúmeras vantagens: muitas funcionalidades, interface gráfica bem evoluída e agradável, resultados impressionantes de editoração, animação e visualização, porém são muito caros. Para OpenGL encontramos Blender [BLENDER] e OpenFX [OPENFX] e para Direct3D encontramos Morfit [MORFIT], todos de domínio público. Para VRML foram encontrados alguns editores para o arquivo no formato VRML, tais como o VrmlPad [VRMLPAD] de domínio privado da ParallelGraphics. Para Java 3D não foram encontrados editores. Para Open Inventor não encontramos um editor 3D de propósito geral, o programador desenvolve seu próprio editor 3D de acordo com a necessidade de sua aplicação usando a biblioteca Open Inventor para realizar tal tarefa. Um exemplo é o Alive [WERNECKE] que é um software de animação de personagens em tempo real.

2.3 Conclusão

Neste capítulo foi apresentado um macro algoritmo de um sistema de simulação baseada em física e foi proposto um macro algoritmo para um sistema de simulação de robôs que incorpora além da modelagem dinâmica a modelagem do controle de movimentação do robô.

Uma vez definido o algoritmo do sistema de simulação, partiu-se para a busca e análise de ferramentas que facilitassem a implementação deste trabalho. Avaliou-se então algumas API's e linguagens de programação destacando-se as vantagens de desvantagens de cada uma delas. Esta busca por ferramentas foi inicialmente norteada pelo desejo de disponibilizar para o usuário um sistema de simulação onde ele pudesse editar o modelo geométrico do robô com um a interface 3D amigável e pela nossa necessidade de manipular os objetos geométricos, criados pelo usuário, através de uma interface de programação com alto nível de abstração. Procurou-se então por modelador geométrico com uma API e que fosse de domínio público. O trabalho inicial foi tentar encontrar este editor 3D, com o qual seria feita a edição rudimentar do modelo geométrico do robô, e que o mesmo disponibilizasse uma interface de programação para a associação deste modelo com o algoritmo da simulação. Os seguintes editores de modelos 3D foram avaliados:

- Blender – editor 3D com API em Phython sobre OpenGL;
- Morfit – editor 3D com API em C++ sobre Direct3D;
- OpenFX – editor 3D sem API sobre OpenGL;

Depois destas pesquisas e análises de várias ferramentas chegou-se a conclusão que era mais vantajoso investir em uma API mais poderosa e em um trabalho futuro incorporar no sistema de simulação um editor 3D amigável para o usuário, pois os editores 3D acadêmicos encontrados possuíam uma interface gráfica com alta curva de aprendizagem, ou não possuíam uma interface de programação adequada aos nossos propósitos, ou em muitos casos os dois. Decidiu-se então procurar prioritariamente pelo requisito interface de programação, já que este requisito possuía maior impacto sobre o projeto e o modelo geométrico do robô não exigia tanta exatidão. Esta procura foi conduzida pela necessidade de se manipular objetos e não desenhos ou conjunto de vértices, para que, com este grau de abstração fosse possível concentrar os esforços na simulação da dinâmica e do controle do robô.

Tendo feito anteriormente um breve estudo sobre Direct3D, Java 3D, VRML e Open Inventor, decidiu-se utilizar esta última biblioteca e carregar o modelo geométrico do robô através do arquivo de descrição de cena do Open Inventor. Este arquivo pode ser editado com qualquer editor ASCII, desde que sejam seguidas as normas do formato lido pela biblioteca Open Inventor. Neste arquivo foram incluídas também, as informações sobre a lei de controle e de dinâmica do robô.

O capítulo 4 detalha o modelo geométrico da biblioteca Open Inventor e como foram incluídas as informações de controle e dinâmica do robô no arquivo de descrição geométrica do robô ou arquivo de descrição da cena no formato Open Inventor.

3 Modelo de controle e de dinâmica

Neste capítulo são apresentadas as teorias de controle e de dinâmica necessárias à criação do modelo de controle e do modelo de dinâmica do robô. A palavra modelo aqui é usada para dar nome a um conjunto de equações que descrevem o comportamento dinâmico do robô e a lei de controle que atua na movimentação do mesmo.

Para o projeto do sistema de simulação adotou-se uma abstração do modelo de controle e do modelo de dinâmica feita por módulos chamados de Módulo de Controle e Módulo de Dinâmica cuja funcionalidade também é descrita neste capítulo. Os detalhes de implementação destes e outros módulos são descritos no capítulo 5.

Apesar do usuário iniciar sua análise pelo modelo de dinâmica do robô, optou-se por apresentar na ordem: modelo de controle seguido do modelo de dinâmica, por que o Módulo de Controle é o primeiro a atuar após o envio de comandos de movimentação para o robô, conforme pode ser visto na figura 3-1. O Módulo de Controle envia para o Módulo de Dinâmica uma atuação, este por sua vez, responde à atuação calculando a nova posição e velocidade das partes que compõe o robô. O resultado deste cálculo é enviado ao Módulo de Animação que se encarrega de redesenhar o robô em sua nova posição/orientação. A figura 3-1 a seguir mostra a interação entre o Módulo de Controle e o Módulo de Dinâmica.

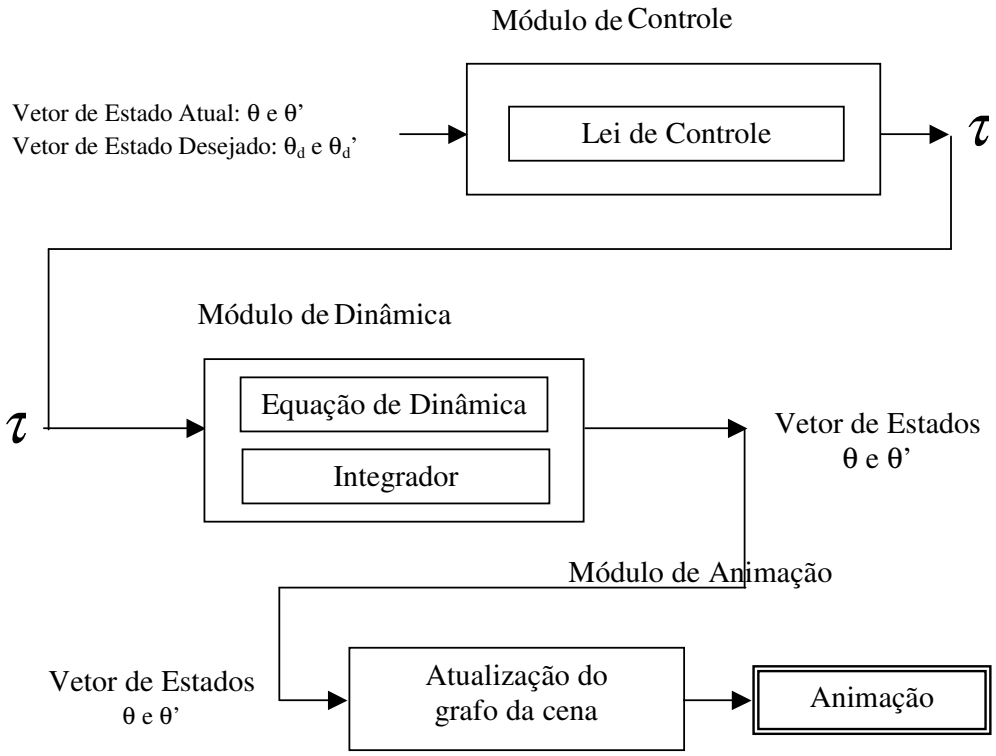


Figura 3-1 - Interação entre o Módulo de Controle e o Módulo de Dinâmica

Uma característica que os módulos de controle e de dinâmica devem apresentar é a capacidade de interpretar tanto as equações de controle como as equações de dinâmica que descrevem o comportamento do robô a ser simulado.

A sessão 3.1 apresenta o modelo de controle e a terminologia de controle, a sessão 3.2 apresenta o modelo de dinâmica e os princípios de dinâmica necessários à obtenção deste modelo e a sessão 3.3 apresenta um exemplo de como obter o modelo dinâmico e de controle para dois sistemas dinâmicos: o pêndulo invertido e um manipulador com dois graus de liberdade.

3.1 O modelo ou lei de controle

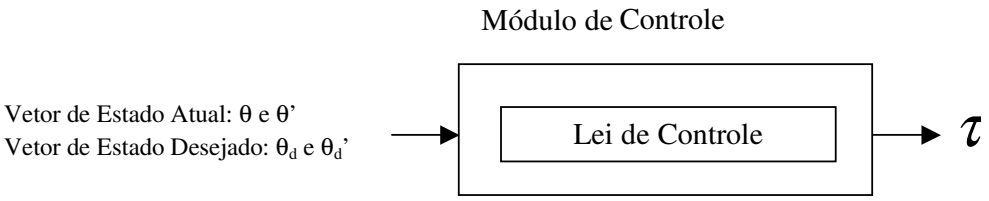


Figura 3-2 - Módulo de Controle

O módulo de controle recebe como entrada um vetor de estado atual do robô (posições e velocidades atuais de cada parte do robô) e um vetor de estado desejado (posições e velocidades desejados de cada parte do robô) e retorna como saída um vetor de força/torque generalizada (τ) que deve ser aplicado às partes do robô para que ele efetue o movimento desejado. Estes vetores de estado atual e desejado são fornecidos pelo usuário respectivamente através do arquivo de descrição do robô, detalhado no capítulo 4, e do arquivo de comandos de movimentação para o robô. O modelo ou a lei de controle que é utilizada pelo módulo de controle também é fornecido pelo usuário através do arquivo de descrição do robô. Esta lei de controle é uma equação que relaciona o vetor de força generalizada (τ) com os vetores de estado atual e estado desejado. A saída deste módulo, o vetor de força generalizada, é passada como entrada para o módulo de dinâmica descrito a seguir na sessão 3.2.

Como o usuário dever fornecer a lei de controle, ou seja, o **modelo de controle** utilizado para governar a movimentação do robô, a sessão 3.1.1 apresenta a terminologia de controle clássico necessária à obtenção deste modelo de controle.

3.1.1 Terminologia da teoria de controle

A maioria dos sistemas de movimentação usa basicamente três métodos para controlar a movimentação, no que diz respeito à variável de interesse do controle: **controle por posição**, **controle por velocidade** e **controle por torque**.

O **controle por posição** atua no sistema fazendo com que o corpo se movimente de uma posição inicial conhecida para uma posição final conhecida.

O **controle por velocidade** atua no sistema fazendo com que o corpo se movimente continuamente em um certo intervalo de tempo ou se movimente de uma posição para outra com uma velocidade pré-definida.

O **controle por torque** controla o torque aplicado pelo atuador na execução de uma tarefa.

Os sistemas de controle podem ser também classificados quanto à forma de medir as variáveis de interesse (posição, velocidade ou torque): **controle de malha aberta** e **controle de malha fechada**.

No **controle de malha aberta** não é feita uma medida da variável de interesse e nem uma atuação na saída do sistema. A saída é função somente da entrada e de alguma transformação que se queira fazer na entrada.

No **controle de malha fechada** a variável de interesse é medida na saída do sistema e comparada com seu valor de entrada desejado para que uma ação corretiva seja tomada e o resultado desejado seja alcançado. A saída é função da entrada e de uma comparação entre a entrada e a saída. O resultado da comparação entre a entrada e a saída, é usado como entrada e é chamado de sinal de realimentação. A forma de usar um sinal de realimentação pode ser classificada em realimentação positiva ou negativa dependendo se o valor da variável de interesse, medida na saída dos sistemas, é somado ou subtraído ao valor da variável de interesse medida na entrada dos sistemas. Este sinal de realimentação pode ser calculado de várias formas: **proporcional**, **derivativo**, **integral,** uma **combinação** dos mesmos ou por **compensação** (*feed forward*).

Controle proporcional: o sinal de realimentação é proporcional a variável controlada.

Controle derivativo: o sinal de realimentação é proporcional à derivada da variável controlada.

Controle integral: o sinal de realimentação é proporcional à integral da variável controlada.

Controle com combinação das características proporcional, derivativa e integral: o sinal de realimentação é proporcional a uma combinação da variável controlada, de sua derivada e de sua integral. A figura 3-3 apresenta um diagrama de blocos de um controlador proporcional integral derivativo (controlador PID).

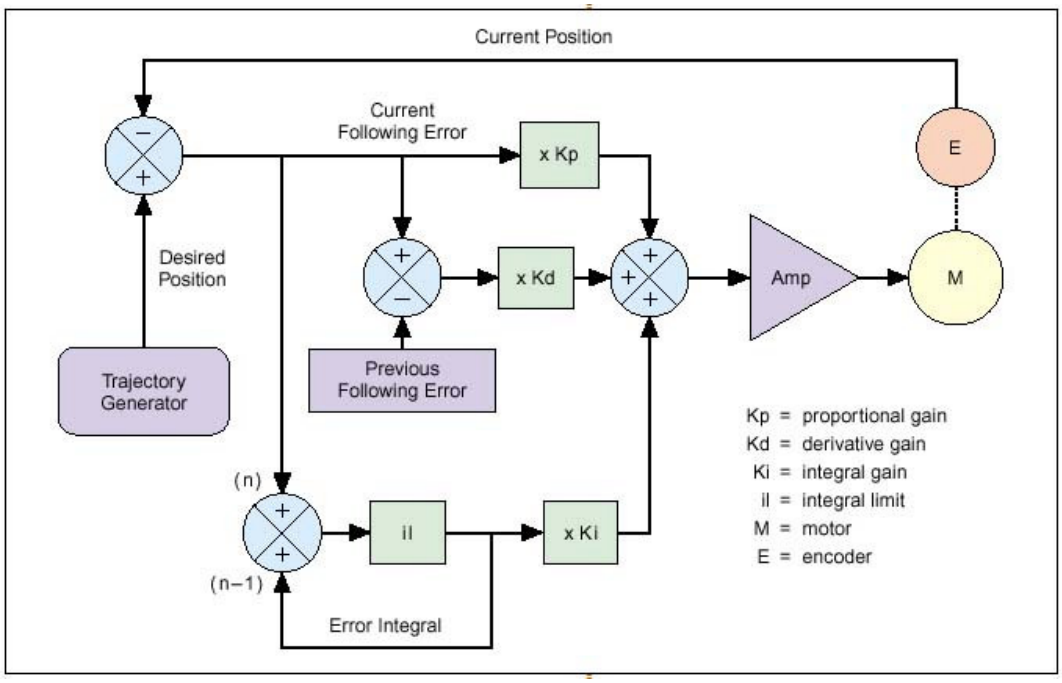


Figura 3-3 – Diagrama de blocos de um controlador PID [NEWPORT]

O **controle por pré-alimentação** (*feed forward*) é utilizado quando o modelo dinâmico do sistema a ser controlado é complexo, ou seja, com muitas equações com alto grau de não linearidade e com equações distintas para cada faixa de trabalho do robô, por exemplo, parado ou em movimento e com ou sem carga. Mas o que é então o controle por pré-alimentação? Ao invés de se tentar embutir na malha fechada de controle todas as variáveis existentes no modelo de dinâmica, utilizam-se elementos compensatórios que já injetam no circuito um sinal relativo a uma característica do sistema, por exemplo, podemos compensar a força de atrito, a gravidade, ou outra grandeza física qualquer. Esta pré-alimentação é feita por um cálculo antecipado do sinal a ser diretamente alimentado. A figura 3-4 apresenta o digrama de blocos de um controlador *feed forward.* Comparando-se as figuras 3-3 e 3-4, podemos notar que a diferença está no sinal retirado do sistema de geração de trajetória (*Trajectory Generator*) que é injetado diretamente na saída. Este sinal tem a função de compensar erros que o controlador PID não consegue.

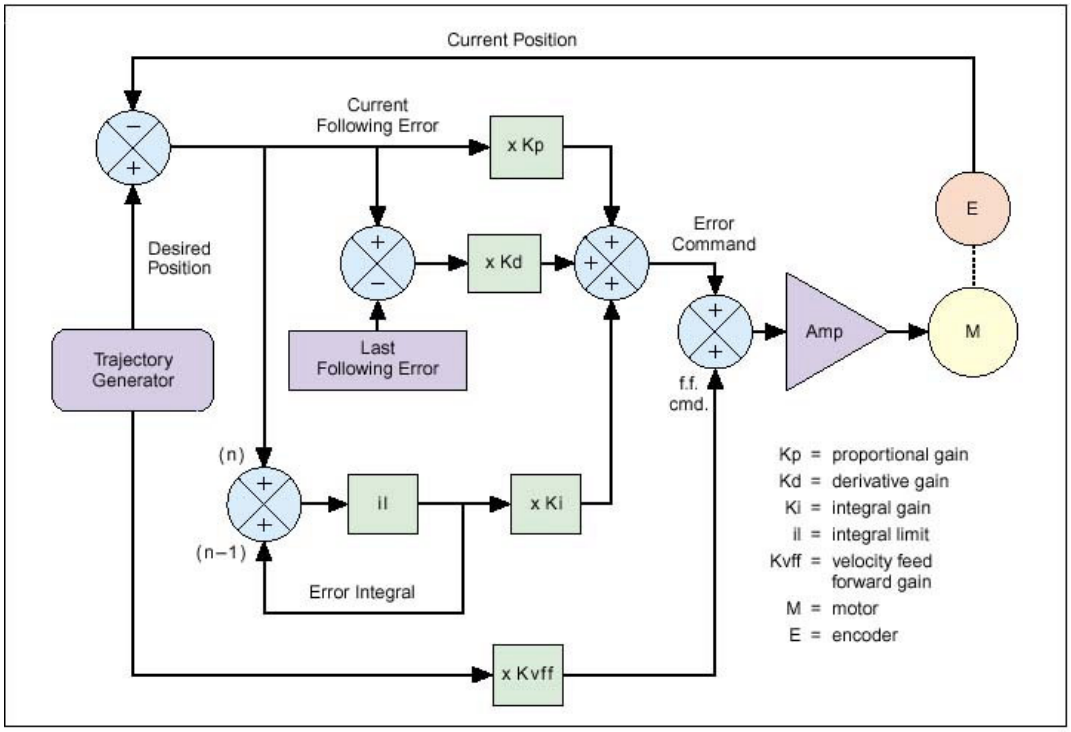


Figura 3-4 Diagrama de blocos de um controlador *Feed Forward* [NEWPORT]

Utilizando um dos tipos de controlador apresentados nesta sessão, o usuário escreve a lei de controle que rege a movimentação do robô.

3.2 O modelo ou equações de dinâmica

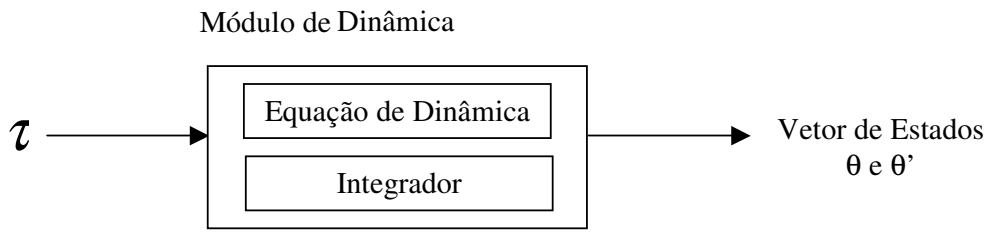


Figura 3-5 - Módulo de Dinâmica

O módulo de dinâmica recebe como entrada o vetor de torque/força generalizada (τ) disponibilizado pelo módulo de controle. De posse deste vetor de força e da equação de dinâmica, fornecida pelo usuário através do arquivo de descrição do robô, o integrador, utilizando um método de integração numérica conhecido, calcula o novo vetor de estado (posição e velocidade). A saída do módulo de dinâmica é passada ao módulo de visualização que se encarrega de atualizar a cena com o robô na nova posição. O integrador calcula os vários vetores de estado intermediários entre a posição atual e a posição desejada passada pelo comando de movimentação do robô. É a partir da utilização de alguns valores contidos no vetor de estado, mais precisamente a posição e a orientação, que se cria a animação.

Como o usuário dever fornecer as equações de dinâmica do robô, ou seja, o **modelo de dinâmica** utilizado para descrever o comportamento dinâmico do robô, a sessão 3.2.1 apresenta as leis fundamentais da dinâmica para partículas e para corpos rígidos, a sessão 3.2.2 apresenta duas abordagens para a realização de simulação de corpos rígidos, a sessão 3.2.3 apresenta os problemas relacionados à colisão entre corpos rígidos, a sessão 3.2.4 apresenta as equações de movimento para robôs manipuladores e a sessão 3.2.5 apresenta como é feita a integração das equações de movimento dos corpos rígidos.

3.2.1 Princípios de dinâmica

Nesta sessão é feita uma breve explicação sobre alguns conceitos de dinâmica de corpos rígidos que servirão como base de apoio ao desenvolvimento deste trabalho.

Inicialmente conceitua-se a dinâmica de uma partícula no espaço, posteriormente estende-se este conceito a um sistema de partículas e a um corpo rígido. Uma partícula é um objeto que possui massa, posição e velocidade, responde a forças, porém não possui extensão espacial [WITKIN 2].

3.2.1.1 Dinâmica de uma partícula

A segunda lei de Newton permite descrever de forma completa a dinâmica de uma partícula. Esta lei pode ser considerada como a definição de força e de massa. Para uma partícula a forma correta da lei é [GOLDSTEIN]:

$$\vec{F} = \frac{d\vec{p}}{dt} \qquad \textbf{Equação 3-1}$$

onde $\vec{F}$ é a força total aplicada à partícula e $\vec{p}$ é o momento linear da partícula.

A definição do momento linear de uma partícula é feita em função da velocidade da mesma sendo dada por:

$$\vec{p} = m\vec{v} \qquad \textbf{Equação 3-2}$$

Sendo que o vetor velocidade é definido por:

$$\vec{v} = \frac{d\vec{r}}{dt} \qquad \textbf{Equação 3-3}$$

Pode-se reescrever a equação 3-1 como:

$$\vec{F} = \frac{d}{dt}(m\vec{v}) \qquad \textbf{Equação 3-4}$$

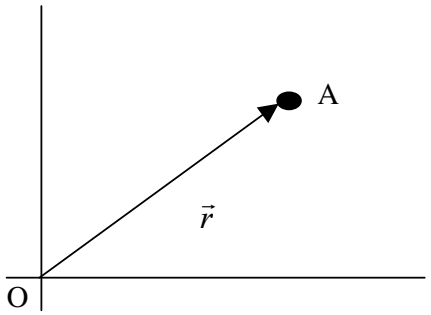
Como neste caso a massa é constante, tem-se:

$$\vec{F} = m\frac{d\vec{v}}{dt} = m\vec{a} \qquad \textbf{Equação 3-5}$$

Pode-se então definir a aceleração da partícula como:

$$\vec{a} = \frac{d\vec{v}}{dt} = \frac{d^2\vec{r}}{dt^2} \qquad \textbf{Equação 3-6}$$

O momento angular da partícula A em torno de um ponto O, representado por $\vec{L}$, é definido como:



$$\vec{L} = \vec{r} \times \vec{p} \qquad \textbf{Equação 3-7}$$

Figura 3-6 - Partícula a uma distância r da origem

Onde $\vec{r}$ é o vetor do ponto O até a partícula A.

O torque ou momento da força da partícula em torno do ponto O, representado por $\vec{\tau}$, é definido como:

$$\vec{\tau} = \vec{r} \times \overrightarrow{F} \qquad \textbf{Equação 3-8}$$

Substituindo $\vec{F}$ na equação 3-8, por sua definição feita equação 3-4, temos:

$$\vec{\tau} = \vec{r} \times \frac{d}{dt}(m\,\vec{v}) \qquad \textbf{Equação 3-9}$$

De acordo com a propriedade:

$$\frac{d}{dt}(\vec{r} \times m\vec{v}) = \frac{d\vec{r}}{dt} \times m\vec{v} + \vec{r} \times \frac{d}{dt}(m\vec{v}) \qquad \textbf{Equação 3-10}$$

Pode-se reescrever a equação 3-9 da seguinte forma:

$$\vec{\tau} = \frac{d}{dt}(\vec{r} \times m\,\vec{v}) = \frac{d\vec{L}}{dt} \qquad \textbf{Equação 3-11}$$

Já que o primeiro termo do lado direito da equação 3-10 é nulo.

Tanto $\vec{\tau}$ quanto $\vec{L}$ são dependentes do ponto em torno do qual são calculados.

Além dos conceitos de força, momento linear, momento angular e torque também é necessário definirmos a energia da partícula. Pode-se começar pela definição do trabalho realizado pela força $\vec{F}$ aplicada à partícula que provoca um deslocamento de um ponto 1 até um ponto 2 como:

$$W_{12} = \int \vec{F} \cdot ds = m \int \frac{d\vec{v}}{dt} \cdot \vec{v} dt = \frac{m}{2} \int \frac{d}{dt}(v^2) dt = \frac{m}{2}(v_2^2 - v_1^2) \quad \textbf{Equação 3-12}$$

Como o valor $mv^2/2$ é a energia cinética da partícula, pode-se concluir que o trabalho realizado pela força $\vec{F}$ para deslocar a partícula do ponto 1 ao ponto 2 é equivalente à variação de energia cinética entre o ponto final e o ponto inicial. A energia cinética da partícula é representada por *T*.

$$W_{12} = T_2 - T_1 \quad \textbf{Equação 3-13}$$

Quando a força aplicada à partícula é proveniente de um campo, o trabalho realizado em um caminho fechado é nulo. Este é o caso da força da gravidade, que age sobre os corpos que estão imersos no campo gravitacional terrestre.

$$\oint \vec{F} \cdot ds = 0 \quad \textbf{Equação 3-14}$$

Isto aconteceria num sistema ideal ou conservativo onde não houvesse forças dissipativas tais como atrito. Como estas forças dissipativas existem, a integral da equação (3-14) nunca será nula.

Pelo teorema de Stokes a equação 3-14 pode ser rescrita como:

$$\nabla \times \vec{F} = 0 \quad \textbf{Equação 3-15}$$

Para que a equação 3-15 seja verdadeira, ou seja, o resultado do produto vetorial seja nulo, $\vec{F}$ deve ser igual ao gradiente de um escalar:

$$\vec{F} = -\nabla V \quad \textbf{Equação 3-16}$$

Onde V representa a energia potencial.

Para um sistema conservativo o trabalho realizado pela força proveniente de um campo, independe do caminho de integração entre os pontos 1 e 2, dependendo somente dos pontos inicial e final (1 e 2). Este trabalho é dado por:

$$W_{12} = V_1 - V_2 \quad \textbf{Equação 3-17}$$

Combinando as equações 3-13 e 3-17 temos:

$$T_1 + V_1 = T_2 + V_2 \qquad \textbf{Equação 3-18}$$

Pode-se relacionar às equações 3-1, 3-11 e 3-18 respectivamente aos seguintes teoremas:

1) **Teorema da conservação do momento linear de uma partícula:** Se a força total, $\vec{F}$ aplicada sobre a partícula é zero, então $\dot{\vec{p}} = 0$ e o momento linear é conservado mantendo-se constante.
2) **Teorema da conservação do momento angular de uma partícula:** Se o torque total, $\vec{\tau}$, é zero então $\dot{\vec{L}} = 0$, e o momento angular $\vec{L}$ é conservado.
3) **Teorema da conservação da energia de uma partícula:** Se as forças que atuam na partícula são conservativas, então a energia total da partícula que é equivalente a $T + V$, é conservada.

3.2.1.2 Dinâmica de um sistema de partículas

Para se estender os conceitos anteriores a um sistema de partículas, é necessário diferenciar as forças internas das forças externas aplicadas a este sistema. As forças externas são provenientes de fontes externas ao sistema, já as forças internas são provenientes das forças que uma partícula aplica nas outras e vice-versa.

Pode-se reescrever a segunda lei de Newton para um sistema de partículas da seguinte forma:

$$\sum \vec{F}_{ji} + \vec{F}_i^{(e)} = \dot{\vec{p}}_i \qquad \textbf{Equação 3-19}$$

Onde o primeiro termo do lado esquerdo da equação 3-19 representa a força interna que i-ésima partícula faz sobre a j-ésima partícula. O segundo termo do lado esquerdo da equação 3-19 representa as forças externas.

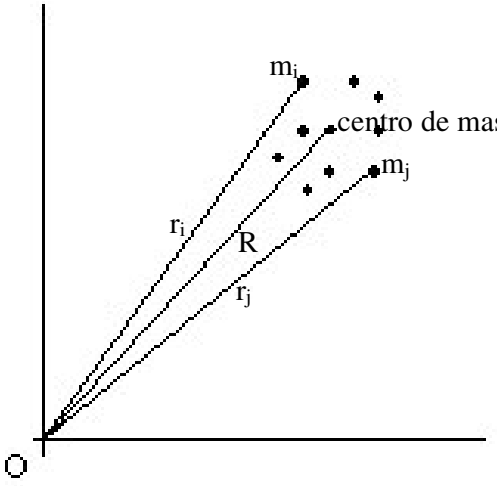
Considerando o caso de forças que obedecem a terceira lei de Newton, a lei da ação e reação, a resultante das forças que duas partículas exercem uma sobre a outra é nula. Reescrevendo a equação 3-19 tem-se:

$$\frac{d^2}{dt^2}\sum_i m_i \vec{r}_i = \sum_i \vec{F}_i^{(e)} + \sum_{\substack{i,j \\ i \neq j}} \vec{F}_{ij} \qquad \textbf{Equação 3-20}$$

Sendo que, o segundo termo do lado direito da equação 3-20 é nulo.

Definindo o centro de massa do sistema de partículas, poder-se reescrever o momento linear, o momento angular, e a energia cinética e potencial para este sistema.

O centro de massa ou centro de gravidade, $\vec{R}$, de um sistema de partículas pode ser definido como sendo um vetor que representa a posição média das massas que compõem este sistema, e é calculado como mostra a equação 3-21.



$$\vec{R} = \frac{\sum m_i \vec{r}_i}{\sum m_i} = \frac{\sum m_i \vec{r}_i}{M} \qquad \textbf{Equação 3-21}$$

Figura 3-7 - O centro de massa de um sistema de partículas

Combinando as equações 3-20 e 3-21 tem-se:

$$M \frac{d^2 \vec{R}}{dt^2} = \sum_i \vec{F}_i^{(e)} \equiv \vec{F}^{(e)} \qquad \textbf{Equação 3-22}$$

De onde podemos concluir que o centro de massa se move como se uma única força externa agisse sobre um corpo de massa *M* concentrada no centro de massa.

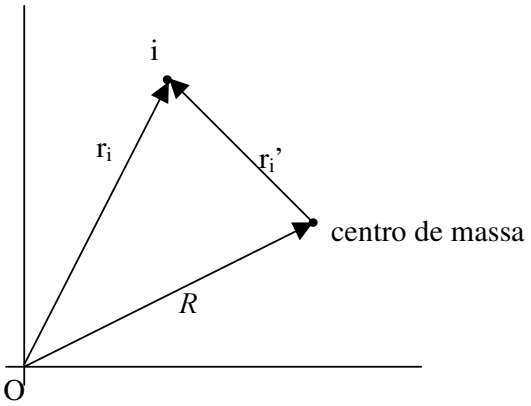
O momento linear do sistema pode ser escrito como:

$$\vec{P} = \sum m_i \frac{d\vec{r}_i}{dt} = M \frac{d\vec{R}}{dt} \qquad \textbf{Equação 3-23}$$

A equação 3-23 mostra que o momento linear do sistema de partículas seria o mesmo se a massa das partículas estivessem concentradas no centro de massa se movendo com ele. Apesar de ser um pouco mais complicado pode-se, por analogia, definir o momento angular total do sistema de partículas por:

$$\vec{L} = \sum_i \vec{r}_i \times \vec{p}_i \qquad \textbf{Equação 3-24}$$

De acordo com a figura 3-8, temos $\vec{R}$ como o vetor da origem até o centro de massa, e $\vec{r}_i'$ como o vetor do centro de massa até a i-ésima partícula.



$$\vec{r}_i = \vec{r}_i' + \vec{R} \qquad \textbf{Equação 3-25}$$

$$\vec{v}_i = \vec{v}_i' + \vec{v}$$

$$\vec{v}' = \frac{d\vec{r}'}{dt}$$

$$\vec{v} = \frac{d\vec{R}}{dt}$$

Figura 3-8 - Vetores para o cálculo do momento angular

Desenvolvendo a equação 3-24, substituindo r_i e p_i , e levando-se em consideração que os termos que contém o fator $\sum m_i \vec{r}_i'$ são nulos, uma vez que estes representam o vetor do centro de massa em relação ao centro de massa, chega-se à seguinte expressão para o momento angular em torno de *O*:

$$\vec{L} = \vec{R} \times M\vec{v} + \sum_i \vec{r}_i' \times \vec{p}_i' \qquad \textbf{Equação 3-26}$$

Com a equação 3-26, pode-se concluir que o momento angular total de um sistema de partícula é momento angular da massa concentrada em relação ao ponto O somado aos momentos angulares de cada partícula em relação ao centro de massa.

A derivada em relação ao tempo de $\vec{L}$, nos dá o torque externo aplicado ao sistema de partícula.

$$\frac{d\vec{L}}{dt} = \tau^{(e)} \qquad \textbf{Equação 3-27}$$

Em termos de energia pode-se escrever a equação da energia cinética do sistema de partículas como:

$$T = \frac{1}{2}\sum_i m_i v_i^2 \qquad \textbf{Equação 3-28}$$

Da mesma forma como se fez para calcular o momento angular, substitui-se v_i por $v + v_i'$ e obtém-se:

$$T = \frac{1}{2} M v^2 + \frac{1}{2} \sum_i m_i v_i'^2 \qquad \textbf{Equação 3-29}$$

A energia cinética também é constituída de duas partes: a primeira é obtida como se toda a massa estivesse concentrada no centro de massa, e a segunda parte é proveniente da movimentação das partículas em torno do centro de massa.

Se as forças internas e externas aplicadas ao sistema de partículas forem provenientes de um campo, a energia potencial para este sistema pode ser definida como:

$$V = \sum_i V_i + \frac{1}{2} \sum_{\substack{i,j \\ i \neq j}} V_{ij} \qquad \textbf{Equação 3-30}$$

O segundo termo do lado direito da equação 3-30, pode variar em alguns sistemas, porém quando se trata de corpos rígidos este termo é constante e é conhecido como energia potencial interna do sistema.

Também para um sistema de partículas vale o teorema de conservação da energia total do sistema, onde $T + V$ é constante.

3.2.1.3 Restrições ao movimento

De acordo com as definições anteriores a solução de problemas de dinâmica parece se resumir a resolver um conjunto de equações diferenciais do tipo:

$$m_i \ddot{\vec{r}}_i = \sum_j \vec{F}_{ji} + \vec{F}_i^{(e)} \qquad \textbf{Equação 3-31}$$

A partir das forças aplicadas às partículas do sistema, a solução da equação 3-31 dá a posição e a velocidade das mesmas. Porém esta forma de visualizar o problema é uma simplificação da realidade física. Na verdade não se está levando em consideração restrições que limitam a movimentação do sistema de partículas. Por exemplo, a própria definição de corpos rígidos como sendo um sistema de partículas onde a distância entre elas é constante já caracteriza uma restrição de movimento. Outro exemplo é as moléculas de um gás, contido em um recipiente, que possuem sua movimentação restringida pelas paredes do recipiente.

Segundo [SPONG] as restrições podem ser classificadas como:

1) Holonômicas: quando descreve como é exatamente a relação entre a posição das partículas e pode ser expressa por equações, do tipo $f(r_1, r_2, r_3...r_n,t) = 0$, que relacionam as coordenadas generalizadas das partículas no tempo.
2) Não holonômicas: restrição que descreve uma relação não exata entre as posições das partículas e pode ser expressa por uma inequação do tipo $f(r_1, r_2, r_3,...,r_n,t) >= 0$. Segundo [GILLESPIE], o termo "não holonômica" significa que as restrições são expressas por relações entre as velocidades generalizadas que não podem ser integradas para produzir os relacionamentos entre as coordenadas generalizadas. Este tipo de restrição pode ser sub-classificado em:
    - Escleronômicas: restrições não holonômicas que não dependem do tempo.
    - Rheonômicas: restrições não holonômicas que dependem do tempo.

Devido à existência destas restrições, a solução dos problemas de dinâmica sai do plano ideal que seria a simples solução de equações diferencias como a equação 3-31, e passa a ter dificuldades tais como: a interdependência entre as coordenadas *ri* e a introdução de novas incógnitas, que são as forças que mantém tais restrições.

Quando tem-se um sistema de N partículas sem restrições dizemos que o sistema apresenta $3N$ graus de liberdade, ou seja, cada partícula possui, por exemplo, movimento livre no eixo x, y e z em coordenadas cartesianas e este movimento é independente do movimento das outras partículas. Se for introduzida alguma restrição neste sistema, as coordenadas das partículas não são mais independentes, pois se relacionam de alguma forma através de equações que expressam estas restrições. A quantidade de graus de liberdade passa a ser $3N$ – (a quantidade de restrições).

A interdependência entre as coordenadas r_i, pode ser resolvida pela introdução do que é conhecido na literatura como *coordenadas generalizadas* $q_1, ..., q_n$ [GOLDSTEIN]. As coordenadas generalizadas são independentes e podem representar posições, ângulos, etc. Considerando um sistema com restrições holonômicas, podemos resolver o problema com coordenadas generalizadas que descrevem melhor (unicamente) a posição e introduzem implicitamente as restrições ao movimento ao invés de coordenadas cartesianas, que podem não representar unicamente a posição dos copos. As coordenadas generalizadas encapsulam as restrições holonômicas do sistema [GILLESPIE]. Por este motivo pode parecer mais fácil resolver os problemas de

dinâmica pelo método de Lagrange (apresentado a seguir), porém é muito difícil ou às vezes impossível encontrar a função paramétrica que descreve as restrições ao movimento [WITKIN 1] em função das coordenadas generalizadas. Segundo [GOLDSTEIN], nos casos onde o princípio do Trabalho Virtual é válido, ou seja, nos casos onde o trabalho realizado pelas forças de restrição é nulo, é possível analisar a dinâmica de um corpo sem a necessidade de se conhecer as tais forças de restrição.

A análise da dinâmica de um corpo pode ser feita segundo algumas formulações já estabelecidas há algum tempo pelos físicos e matemáticos do passado, tais como Lagrange-Euler e Newton-Euler, conforme descrito a seguir:

Lagrange-Euler: este método faz a análise dinâmica do corpo a partir da diferença entre a energia cinética e a energia potencial. Neste caso não há necessidade de se conhecer todas as forças que atuam no corpo [CRAIG, SPONG].

Newton-Euler: este método faz a análise dinâmica do corpo a partir de equações que descrevem os deslocamentos lineares e angulares [CRAIG, SPONG].

As formulações de **Lagrange-Euler** e **Newton-Euler** são também conhecidas respectivamente como um método baseado em energia e método vetorial para análise da dinâmica de corpos rígidos.

3.2.1.4 Dinâmica de um corpo rígido

A localização de uma partícula no espaço para um dado instante de tempo *t* pode ser descrita por um vetor $\vec{x}(t)$, o qual descreve a translação da partícula a partir da origem do sistema de referência. A velocidade desta partícula pode ser definida por $\vec{v}(t) = \dot{\vec{x}}(t)$. No caso de um corpo rígido não é tão fácil assim, pois além da translação deve-se considerar a rotação do corpo. Sendo assim para localizar um corpo rígido no espaço além do vetor $\vec{x}(t)$, que fornece a translação, faz-se uso da matriz *R(t)*, que fornece a rotação deste corpo. Segundo [BARAF] $\vec{x}(t)$ e *R(t)* são as variáveis espaciais de um corpo rígido.

Diferentemente de uma partícula um corpo rígido ocupa um volume no espaço com uma determinada forma ou geometria. Para descrever a geometria do corpo faz-se uso de um sistema de coordenadas do objeto. O sistema de coordenadas do objeto é um sistema de referência ou eixos de coordenadas cartesianas cuja origem é fixada no centro de massa do objeto. Por motivos de simplificação, pode-se considerar que o

centro de massa de um corpo está localizado em seu centro geométrico. Como o corpo é rígido, a descrição de sua geometria em relação a um sistema de referência fixado no próprio corpo não muda. O **sistema de coordenadas do objeto** também é chamado de **sistema de referência local** e além do sistema de coordenadas do objeto tem-se também o **sistema de coordenadas do mundo**, que também é chamado de **sistema de referência global**. A figura 3-9 ilustra o sistema de coordenadas local e global.

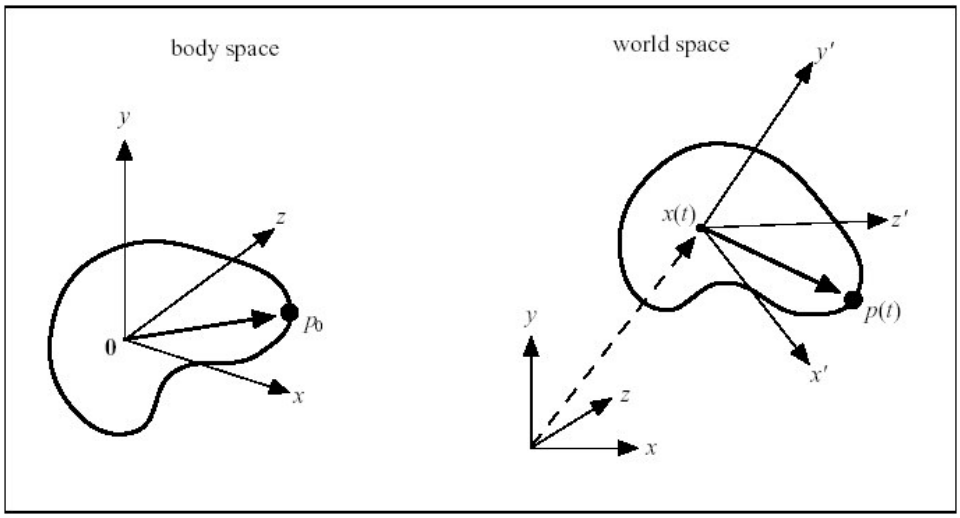


Figura 3-9 - Sistema de coordenadas do objeto e sistema de coordenadas do mundo[BARAF]

Considerando que *R(t)* especifica a rotação do corpo sobre o seu centro de massa em relação às coordenadas do mundo, de acordo com a figura 3-9 tem-se que:

$\vec{x}(t)$	é o centro de massa do corpo descrito no sistema de coordenadas do mundo;
$\vec{x}'(t) = R(t)\vec{x}$	é o eixo x do corpo descrito no sistema de coordenadas do mundo;
$\vec{y}'(t) = R(t)\vec{y}$	é o eixo y do corpo descrito no sistema de coordenadas do mundo;
$\vec{z}'(t) = R(t)\vec{z}$	é o eixo z do corpo descrito no sistema de coordenadas do mundo;
$\vec{p}(t) = R(t)p_0 + \vec{x}(t)$	é a posição de um ponto arbitrário p_0 do corpo descrito no sistema de coordenadas do mundo;

Como os vetores $\vec{x}, \vec{y}, e\ \vec{z}$ descritos no sistema de coordenadas do objeto (figura 3-9) podem ser considerados respectivamente iguais à [1 0 0], [0 1 0] e [0 1 0] temos que no instante de tempo *t* estes vetores têm a direção

$\vec{x}' = R(t)\begin{bmatrix}1\\0\\0\end{bmatrix}$ $\vec{y}' = R(t)\begin{bmatrix}0\\1\\0\end{bmatrix}$ $\vec{z}' = R(t)\begin{bmatrix}0\\0\\1\end{bmatrix}$ no sistema de coordenadas do mundo.

Os resultados das multiplicações acima produzem vetores que são iguais a primeira coluna da matriz *R(t)*, segunda coluna da matriz *R(t)* e terceira coluna da matriz *R(t)*, respectivamente. Em resumo, a matriz *R(t)* dá, em sua primeira coluna, a direção que o eixo x do sistema de coordenadas do corpo possui quando é convertido para o sistema de coordenadas do mundo. O mesmo vale para a segunda e terceira coluna de *R(t)*, por isso dizemos que a localização de um corpo rígido é descrito por uma posição (um vetor $\vec{x}(t)$) e uma orientação (uma matriz *R(t)*).

Pode-se agora, definir a velocidade linear e angular em função de $\vec{x}(t)$ e *R(t)*. Analisando-se separadamente o movimento de translação e o movimento de rotação do corpo rígido, tem-se que a velocidade linear do corpo é dada por $\vec{v}(t) = \dot{\vec{x}}(t)$ para uma translação pura, ou seja, é a velocidade de translação do centro de massa do corpo. Considerando agora que existe somente uma rotação em relação a um eixo qualquer que passe pelo centro de massa do corpo tem-se $\vec{\omega}(t)$ como o vetor que representa a velocidade angular do corpo, onde a direção de $\vec{\omega}(t)$ é a mesma do eixo sobre o qual o corpo está girando e o módulo de $\vec{\omega}(t)$ é a intensidade ou o quão rápido o corpo está girando. O vetor $\vec{\omega}(t)$ é chamado de velocidade angular do corpo. A figura 3-10 mostra os vetores de velocidade linear e angular do corpo rígido.

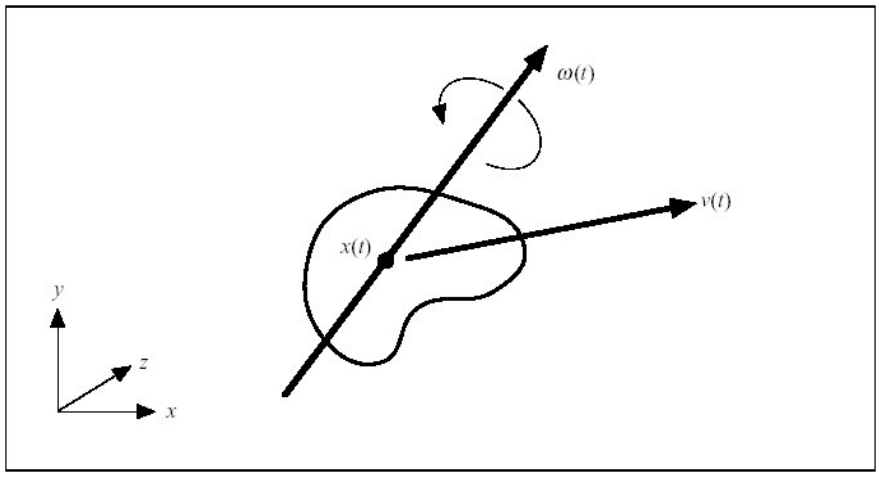


Figura 3-10 - Vetores de velocidade linear e angular do corpo rígido [BARAF]

A velocidade linear tem relação direta com a posição $\vec{v}(t) = \frac{d}{dt}\vec{x}(t) = \dot{\vec{x}}(t)$, já $R(t)$ não possui a mesma relação com $\vec{\omega}(t)$, ou seja, $\dot{R}(t)$ não é $\vec{\omega}(t)$, pois $\dot{R}(t)$ é uma matriz e $\vec{\omega}(t)$ é um vetor.

Uma relação entre $\dot{R}(t)$ e $\vec{\omega}(t)$ é detalhadamente demonstrada no capítulo 4.8 em [GOLDSTEIN]. Para simplificarmos temos:

$$\frac{d}{dt}R(t) = \dot{R}(t) = \tilde{\omega}(t)R(t)$$

onde $\tilde{\omega}(t)$ é uma matriz construída pelos componentes *x*, *y* e *z* de $\vec{\omega}(t)$ da seguinte forma:

$$\tilde{\omega}(t) = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix}$$

Para a análise dinâmica de corpos rígidos partiu-se da definição da dinâmica para uma partícula, na sessão 3.2.1.1; posteriormente analisou-se um sistema de partículas, na sessão 3.2.1.2 e nesta sessão adotou-se a simplificação que um corpo rígido é um sistema de partículas cuja distância entre as partículas é constante. Partindo-se desta simplificação tem-se que as variáveis dinâmicas importantes à análise do corpo são:

$\vec{p}(t) = R(t)p_0 + \vec{x}(t)$ — Posição de um ponto arbitrário p_0 do corpo descrito no sistema de coordenadas do mundo;

I_{body}	Matriz de inércia do corpo descrita em relação ao sistema de coordenadas do corpo. Esta matriz é constante;
$I(t) = R(t) I_{body} R(t)^T$	Tensor de inércia;
$\vec{P}(t) = M\vec{v}(t)$	Momento linear;
$\vec{L}(t) = I(t)\vec{\omega}(t)$	Momento angular;
$\vec{F}(t) = \dot{\vec{P}}(t) = M\frac{d}{dt}\vec{v}(t)$	Força;
$\vec{\tau}(t) = \dot{\vec{L}}(t)$	Torque.

3.2.2 Simulação de dinâmica de corpos rígidos

Como a simulação da dinâmica de sistemas mecânicos pode variar desde a modelagem do problema por partículas até por sistemas de corpos rígidos ou flexíveis, deve ficar claro que a modelagem por sistemas de partículas, sem a restrição de distância constante entre as partículas, é utilizada nos casos onde se quer estudar o comportamento de fluidos, gases e até mesmo de corpos flexíveis e elásticos (sistemas de partículas acopladas por molas) [WITKIN 3]. Já a modelagem por corpos rígidos é mais adequada a sistemas mecânicos onde os corpos em estudo não sofrem deformação.

A expressão "corpos rígidos" é usada aqui para ressaltar que, durante a simulação da movimentação destes corpos, a sua forma não é alterada, ou seja, os corpos no escopo deste trabalho não sofrem deformação. Deste ponto do texto em diante todos os corpos são considerados rígidos.

A tabela a seguir apresenta as variáveis de estado que compõem uma simulação de partículas e uma simulação de um corpo rígido sem restrições de movimento

Partícula	Corpo Rígido
$Y(0) = \begin{pmatrix} \vec{x}(0) \\ \vec{v}(0) \end{pmatrix}$	$Y(0) = \begin{pmatrix} \vec{x}(0) \\ R(0) \\ \vec{P}(0) \\ \vec{L}(0) \end{pmatrix}$
Condição inicial *Y(0)* contendo: a posição e a velocidade inicial da partícula.	Condição inicial *Y(0)* contendo: a posição, a orientação, o momento linear e o momento angular inicial do corpo rígido.

$\frac{d}{dt}Y(t) = \frac{d}{dt}\begin{pmatrix}\vec{x}(t)\\ \vec{v}(t)\end{pmatrix} = \begin{pmatrix}\vec{v}(t)\\ \vec{F}(t)/m\end{pmatrix}$	$\frac{d}{dt}Y(t) = \begin{pmatrix}\frac{d}{dt}\vec{x}(t)\\ \frac{d}{dt}R(t)\\ \frac{d}{dt}\vec{P}(t)\\ \frac{d}{dt}\vec{L}(t)\end{pmatrix} = \begin{pmatrix}\vec{v}(t)\\ \tilde{\omega}(t)R(t)\\ \vec{F}(t)\\ \vec{\tau}(t)\end{pmatrix}$
Vetor de estado *Y(t)* contendo: a derivada da posição e a derivada da velocidade da partícula no instante de tempo *t*.	Vetor de estado *Y(t)* contendo: a derivada da posição, a derivada da orientação, a derivada do momento angular e a derivada do momento linear no instante de tempo *t*.

Tabela 3-1 - Vetor de estados para simulação de partículas e de corpos rígidos

A simulação de dinâmica de corpos rígidos pode ser com ou sem restrição.

- **Simulação de dinâmica de corpos rígidos sem restrição**: este tipo de simulação executa o cálculo do estado dinâmico baseando-se nos conceitos descritos na sessão 3.2.1.4 e engloba os casos onde o corpo em estudo está livre de restrições ao movimento, por exemplo, um corpo em queda livre.
- **Simulação de dinâmica de corpos rígidos com restrição**: este tipo de simulação também executa o cálculo do estado dinâmico baseando-se nos conceitos descritos na sessão 3.2.1.4, porém, além disso, leva em consideração outros elementos que restringem o movimento do corpo. Estes elementos são: atrito, forças de contato, colisão entre corpos ou relações de ligação entre corpos (sistema de corpos articulados), entre outros. Para realizar a simulação, estes elementos podem ser modelados individualmente e utilizados para o cálculo do estado dinâmico *on-line*, ou podemos calcular o estado dinâmico *off-line*, fazendo uso de uma equação diferencial explícita previamente obtida a partir das formulações Lagrange-Euler ou Newton-Euler, que englobe tais elementos que restringem o movimento.

O cálculo do estado dinâmico em ambos os casos é auxiliado por um método numérico iterativo de integração, Runge Kutta ou Euler, uma vez que sistemas complexos geralmente são descritos por equações diferenciais que devem ser integradas para se obter sua solução.

Para simular a dinâmica de corpos rígidos com restrições pode-se adotar uma das duas abordagens: *on-line* ou *off-line*.

Na abordagem *off-line* a equação de dinâmica que descreve o movimento do corpo pode ser gerada automaticamente ou fornecida pelo usuário. Em todos os dois casos é possível utilizar uma das formulações já apresentadas, Lagrange-Euler ou Newton-Euler, para obter a equação de dinâmica do corpo. Utilizando-se uma destas formulações é possível escrever as equações de dinâmica embutido na própria equação algumas restrições de movimento. Escrevendo estas equações em função de coordenadas generalizadas, é possível introduzir, por exemplo, as restrições de geometria, de rigidez e de colisão entre as partes que compõe o corpo rígido. De posse da equação na forma explícita e das condições iniciais do sistema, um método de integração numérica pode ser usado para resolvê-la. A partir das condições iniciais e de uma atuação fornecida, podemos calcular os próximos valores para equação de dinâmica que descreve o movimento do corpo. Em resumo para a abordagem *off-line* existe um conjunto de equações diferenciais previamente determinado, ou seja, definido antes da simulação iniciar, que é utilizado para o cálculo do novo estado dinâmico.

Na abordagem *on-line* não se possui a equação que descreve a dinâmica do corpo. É feita uma descrição do corpo e do relacionamento entre suas partes e entre outros corpos que fazem parte da simulação, e dependendo destas relações e de alguns eventos que ocorrem durante o ciclo, os cálculos de dinâmica durante um ciclo da simulação podem ser diferentes dos cálculos em outro ciclo. Por exemplo, durante um ciclo da simulação uma força de atrito pode estar atuando e em outro não. Durante um ciclo deste tipo de simulação geralmente é preciso fazer o somatório das forças que atuam nos corpos para obter a força resultante que atua no centro de massa de cada corpo. De posse desta força resultante, calcula-se a aceleração que age no centro de massa de cada corpo. Utilizando-se um método de integração numérica calcula-se a velocidade e a posição do corpo após um determinado instante de tempo posterior a aplicação das forças. A partir da força resultante sobre o corpo e da massa do corpo, extrai-se as grandezas lineares (velocidade e posição), em seguida usando o mesmo procedimento a partir dos torques aplicados ao corpo e do momento de inércia do corpo, extrai-se as grandezas angulares (velocidade e posição). Não se pode esquecer que as forças aplicadas ao corpo em direções que não passam no centro de massa também geram torques neste corpo. Além destes cálculos, que estão baseados nas equações

apresentadas na tabela 3-1 para corpos rígidos, uma série de verificações como, por exemplo, contato, colisão, e interpenetração entre corpos, são realizadas a cada ciclo da simulação para todos os todos os corpos, o que pode tornar a simulação muito lenta.

Comparando estas duas abordagens podemos destacar que a abordagem *on-line* tem a vantagem de estar preparada para tratar novos eventos durante a simulação e tem as desvantagens de sempre executar várias verificações a fim de identificar se um determinado evento ocorreu, por isso é mais lenta e potencialmente imprecisa. Já a abordagem off-line tem a desvantagem de não poder, em tempo de execução da simulação, se adequar a um evento não modelado pelo conjunto de equações de dinâmica estabelecido no início da simulação, porém tem as vantagens de não ter que efetuar muitas verificações e tem a tendência de ser mais precisa, pois executa uma seqüência menor de cálculos.

Bart [BARENBURG], em seu trabalho, desenvolveu uma biblioteca de classes em C++ para realizar os cálculos de dinâmica na forma *on-line*. Ele criou vários tipos de junções para representar corpos articulados e escreveu equações para o cálculo das restrições de geometria, de rigidez e de colisão dependendo do tipo de ligação que os corpos articulados possuíam entre si. Após o cálculo da nova posição e velocidade, um gerenciador de restrições calcula: a força, o torque ou o impulso que deve ser aplicado ao corpo para que, no novo cálculo de posição e velocidade, as restrições sejam satisfeitas.

Quando se trata de dinâmica de corpos rígidos, deve-se ter em mente que os movimentos possuem restrições, por isso além de se fazer com que os corpos rígidos obedeçam as Leis de Newton é necessário se preocupar com os impedimentos aos movimentos, por exemplo, devido à geometria destes corpos. Quando simula-se um sistema de corpos rígidos em movimento, é importante a verificação do evento de contato ou colisão entre estes corpos e a respectiva resposta fisicamente correta a este evento. Em se tratando de corpos rígidos, os aspectos inerentes à deformação e interpenetração de objetos não são tratados neste trabalho, apesar de terem sido previstos.

3.2.3 O problema de colisão

No tratamento do evento de colisão entre corpos deve ficar claro existem duas tarefas distintas [MOORE]: a primeira é detectar que a colisão ocorreu, e a segunda é responder à colisão. A detecção da colisão é essencialmente um problema de cinemática, que envolve o conhecimento das relações de posicionamento entre os objetos no mundo, já a resposta à colisão é um problema de dinâmica que envolve antecipar o comportamento dos objetos de acordo as leis da física [MOORE].

O problema de detecção de colisão entre objetos em movimento é fundamental para simulações que necessitem ser fisicamente realísticas. Este problema é comum a várias áreas de estudo, tais como robótica, projeto assistido por computador, computação gráfica e geometria computacional. A detecção de colisão é a ação de determinar quando um objeto penetra outro objeto. Dependendo da complexidade da forma e da quantidade de objetos envolvidos isto pode ser uma tarefa de alto custo computacional.

Existem várias formas de realizar detecção de colisão e alguns algoritmos já implementados e testados [GAO]. O apêndice C traz mais detalhes sobre estes algoritmos.

A resposta, fisicamente coerente, ao evento de colisão entre dois corpos rígidos é essencialmente um problema de dinâmica. Por motivos de simplificação a resposta à colisão pode ser simplesmente informar ao sistema de animação ou de controle de movimentação que ocorreu a colisão passando a responsabilidade de tratamento deste evento para estes sistemas. Em uma modelagem mais elaborada, quando a dinâmica é levada em consideração para a geração do movimento, é possível, após a detecção da colisão, o sistema apresentar um comportamento coerente com a colisão ocorrida. Para isto, o cálculo das velocidades lineares e angulares deve ser feito obedecendo à lei de conservação do momento linear e angular. Aspectos de elasticidade dos corpos também devem se levados em consideração para se determinar o quanto de energia cinética é perdida durante a colisão. Existem algumas abordagens que modelam a resposta à colisão através do cálculo do impulso durante a colisão ou através da inserção temporária de molas entre os corpos que estão em contato.

Impulso – Este método é ideal para situação de choques violentos, porém é mal comportado para o caso de choques suaves.

Molas – mais genérico e mais fácil de programar, porém necessita de um intervalo de tempo muito pequeno para ser exato o que significa um maior número de iterações. Este método não produz uma boa visualização do movimento no caso de choques violentos, já no caso de choques suaves ou no caso de um corpo repousando sobre outro produz ótimos resultados [MOORE].

Tanto a detecção da colisão quanto a resposta à mesma foram deixadas como um módulo a ser implementado num trabalho futuro.

3.2.4 Equações de movimento para robôs

Para a obtenção das equações de movimento dos robôs pode-se analisar a dinâmica do sistema segundo a formulação de Lagrange-Euler (energia) ou Newton-Euler (vetorial). Segundo a formulação de Lagrange-Euler, as equações de movimento, para os robôs manipuladores, são dadas por:

$$\sum_j d_{kj}(q)\ddot{q}_j + \sum_{i,j} c_{ijk}(q)\dot{q}_i\dot{q}_j + \phi_k(q) = \tau_k \quad , \quad k = 1,\ldots,n \qquad \textbf{Equação 3-32}$$

Onde cada termo do lado esquerdo da equação tem um significado físico. O primeiro termo desta equação possui uma derivada de segunda ordem da coordenada generalizada e está relacionado à inércia. O segundo termo possui um produto entre as derivadas de primeira ordem da coordenada generalizada. Quando i=j o termo é associado ao efeito centrífugo, quando j≠i o termo é associado ao efeito Coriolis. O terceiro termo está associado à energia potencial. O termo do lado direito representa a força generalizada (força ou torque) à qual o manipulador é submetido. O indexador *k* numera cada coordenada generalizada, ou seja, a equação 3-32 descreve um conjunto de *n* equações.

Estas equações podem ser escritas a partir das formulações Lagrange-Euler ou Newton-Euler. A tabela a seguir mostra uma comparação do custo computacional dos métodos **Lagrange-Euler** e **Newton-Euler** para a execução da dinâmica inversa do manipulador PUMA 560. A coluna mais à esquerda mostra custo computacional em função do número de somas e multiplicações. *N* representa a quantidade de graus de liberdade do PUMA 560.

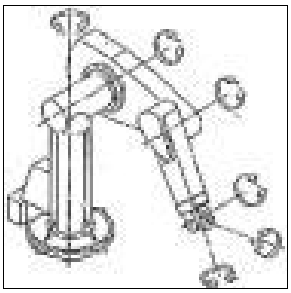


Figura 3-11 Manipulador PUMA 560

Método	Multiplicações	Adições	Para *N*=6	
			Mul	Ad
Lagrange-Euler	$32\frac{1}{2}n^4+86\frac{5}{12}n^3+171\frac{1}{4}n^2+53\frac{1}{3}n-128$	$25n^4+66\frac{1}{3}n^3+129\frac{1}{2}n^2+42\frac{1}{3}n-96$	66271	51548
Newton-Euler Recursivo	150n – 48	131n – 48	852	738

Tabela 3-2- Comparação do custo computacional para cálculo da dinâmica inversa [CORKE]

A tabela a seguir mostra as vantagens e desvantagens de um método em relação ao outro [GILLESPIE] no que diz respeito ao número de variáveis envolvidas, onde:

$\nu \rightarrow$ número de corpos;

$M \rightarrow$ número de restrições holonômicas;

$m \rightarrow$ número de restrições não holonômicas;

$n = 6\nu - M \rightarrow$ número de coordenadas generalizadas;

$\rho = n - m \rightarrow$ número de velocidades generalizadas independentes.

	Newton-Euler	Lagrange-Euler
Variáveis de Configuração	Coordenadas de posição e orientação $(x, y, z, \theta_1, \theta_2, \theta_3)_i$, (i = 1,...,ν)	Coordenadas generalizadas q_i, (i = 1,...,ν)
Variáveis de Movimento	Derivada das coordenadas de posição e orientação $(x, y, z, \theta_1, \theta_2, \theta_3)_i$, (i = 1,...,ν)	Coordenadas generalizadas q_i, (i = 1,...,ν)
Número de Restrições	M + m	m

Tabela 3-3 Comparativo entre os métodos de obtenção das equações de dinâmica [GILLESPIE]

Reescrevendo as equações de movimento 3-32, na forma matricial, tem-se:

$$D(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = \tau \qquad \textbf{Equação 3-33}$$

Para resolver a equação 3-33 tem-se que:

$$D(q)\ddot{q} = \tau - C(q,\dot{q})\dot{q} - g(q)$$

$$\ddot{q} = D^{-1}(q)(\tau - C(q,\dot{q})\dot{q} - g(q)) \qquad \textbf{Equação 3-34}$$

$$\dot{q} = \int D^{-1}(q)(\tau - C(q,\dot{q})\dot{q} - g(q))$$

$$q = \int \dot{q}$$

Para resolver a equação 3-34 faz-se uso de um método de integração conforme apresentado na sessão 3.2.5 a seguir.

3.2.5 Integração das equações de dinâmica

Segundo [BOURG] pode-se considerar que uma simulação é uma **simulação em tempo real** quando esta possui uma base de tempo própria que permita que sejam mantidas a seqüência dos acontecimentos e a relação de tempo entre eles, e quando executa os cálculos do estado dos objetos "*on the fly"* (durante a simulação). Em alguns casos o cálculo do estado dos objetos durante a simulação permite que decisões sejam tomadas e respostas diferentes sejam apresentadas para cada estado. Segundo esta definição pode-se classificar a simulação implementada neste trabalho como uma simulação em tempo real.

Para gerar a animação do robô faz-se uso do estado dinâmico calculado a cada ciclo da simulação. O estado dinâmico do robô é calculado durante a simulação, ou seja, o valor da posição e da velocidade, de cada parte do robô, é calculado em tempo de execução da simulação, nenhum destes valores são pré-calculados. Estes cálculos são realizados com o auxílio de um método de integração numérica como os apresentados a seguir.

No contexto deste trabalho, o estado dinâmico do robô deve estar de acordo com a atuação (força/torque) que o sistema de controle gerou a partir do comando de movimentação enviado ao robô. Para calcular o estado dinâmico do robô faz-se uso de um método de integração numérica para se encontrar a solução aproximada das equações diferenciais ordinárias que descrevem o comportamento dinâmico do robô.

Independente da abordagem, *on-line* ou *off-line*, utilizada para realizar a simulação da dinâmica de um sistema físico qualquer, sempre existe uma etapa que é a integração numérica das equações de movimento do sistema. Neste trabalho os métodos de integração numérica são utilizados para resolver as equações de dinâmica do robô, fornecidas pelo usuário na forma da equação 3-34 apresentada anteriormente. Em seguida é feita uma breve descrição de dois métodos de integração numérica que são muito utilizados: **Euler** e **Runge Kutta.**

3.2.5.1 O Método de Euler

O método de Euler é o de mais simples explicação e implementação, por isso começa-se por ele. Este método é baseado na expansão da série de Taylor que diz que é possível aproximar o valor da função em um ponto conhecendo-se algo sobre a função e suas derivadas em outros pontos. Esta aproximação é expressa série polinomial infinita dada por:

$$y(x+\Delta x) = y(x) + (\Delta x)y'(x) + \left[\frac{(\Delta x)^2}{2!}\right]y''(x) + \left[\frac{(\Delta x)^3}{3!}\right]y'''(x) + \ldots \qquad \textbf{Equação 3-35}$$

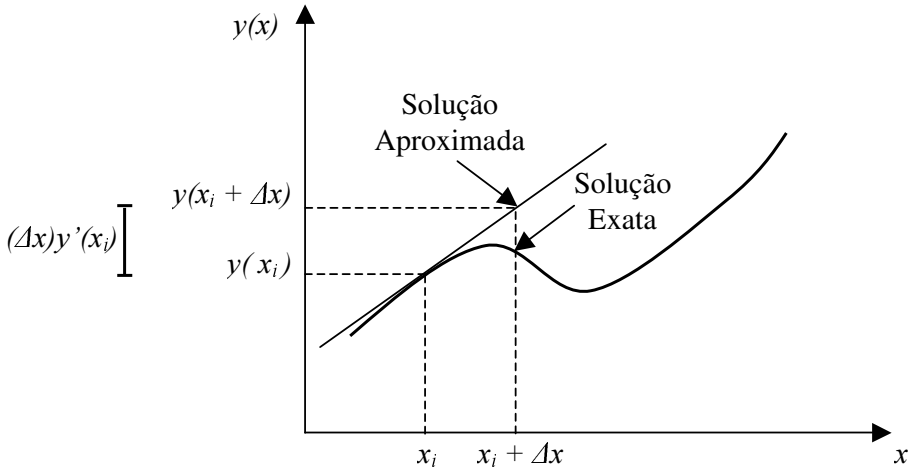
onde:

y	é uma função de x;
$(x+\Delta x)$	é o novo valor de x para onde queremos aproximar y;
y'	é a primeira derivada de y;
y''	é a segunda derivada de y; e assim por diante.

Geralmente trunca-se esta série no segundo termo do lado direito da igualdade e a ausência dos demais termo é considera como erro de aproximação de segunda ordem $O((\Delta x)^2)$.

$$y(x+\Delta x) = y(x) + (\Delta x)y'(x) + O(\Delta x^2) \qquad \textbf{Equação 3-36}$$

A figura 3-37 apresenta um exemplo gráfico da solução da função *y(x)* no ponto $x_i + \Delta x$.



Equação 3-37 Representação gráfica do método de Euler

Este método apresenta uma boa convergência para um valor pequeno de passo de integração *(Δx)*, quanto menor o passo menor será o erro de convergência, porém para uma simulação em que se deseja apresentar os resultados através de uma animação, um valor muito pequeno de passo pode deixar a animação muito lenta, com o efeito indesejado de câmera lenta.

3.2.5.2 O Método de Runge-Kutta

Outro método de integração também bastante difundido é o método de Runge Kutta, que é mais sofisticado que o anterior, pois apresenta melhor convergência melhor. Este método estende a idéia do método de Euler calculando os valores intermediários da derivada de *Y(x)* no intervalo *Δx* e combinando estes valores de forma a anular o erro ocasionado pelo truncamento da série de Taylor. A quantidade de valores intermediários dá nome aos métodos Runge-Kutta de Ordem Dois e Runge-Kutta de Quarta Ordem.

O método Runge-Kutta de Segunda Ordem simula a execução de dois passos do método de Euler em um único passo. Considerando que a derivada de *y* em relação à *x* seja uma função de *x* e *y* tem-se:

$$y'(x) = f_y(x, y)$$

$$K_1 = \Delta x\, f_y(x_i, y)$$

$$K_2 = \Delta x\, f_y(x_i + \frac{\Delta x}{2}, y + \frac{K_1}{2})$$

$$y(x_i + \Delta x) = y(x_i) + K_2 + O(\Delta x^3)$$

Este método também é conhecido como método do ponto do meio, pois usa o valor da derivada da função *f(x)* no ponto central do intervalo *Δx,* como o valor da derivada de *f(x)* em todo o intervalo *Δx*. A figura 3-12 mostra graficamente este conceito.

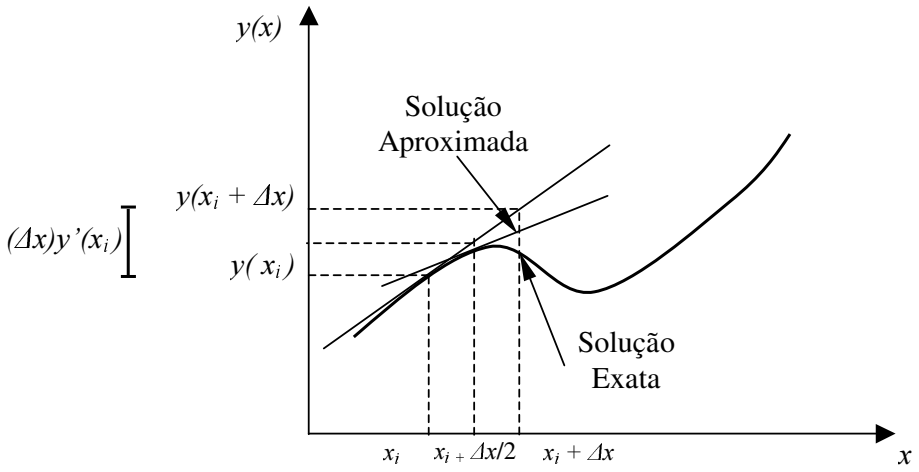


Figura 3-12 Representação gráfica do método de Runge-Kutta de Segunda Ordem

O método Runge-Kutta de Quarta Ordem simula a execução de quarto passos do método de Euler em um único passo.

$$y'(x) = f_y(x, y)$$

$$K_1 = \Delta x\, f_y(x_i, y)$$

$$K_2 = \Delta x\, f_y(x_i + \frac{\Delta x}{2}, y + \frac{K_1}{2})$$

$$K_3 = \Delta x\, f_y(x_i + \frac{\Delta x}{2}, y + \frac{K_2}{2})$$

$$K_4 = \Delta x\, f_y(x_i + \Delta x, y + K_3)$$

$$y(x_i + \Delta x) = y(x_i) + \frac{K_1}{6} + \frac{K_2}{3} + \frac{K_3}{3} + \frac{K_4}{6} + O(\Delta x^5)$$

Usando um dos métodos apresentados acima pode-se resolver as equações de dinâmica do robô, encontrando os novos valores de posição e velocidade (vetor de estado) que o robô deve possuir devido à aplicação da atuação oriunda do módulo de controle. Com estes novos valores de posição e velocidade redesenha-se o robô a cada passo de integração ou um múltiplo do passo de integração gerando com isto a animação.

3.3 Exemplo: Obtenção do modelo de dinâmica e de controle para dois sistemas dinâmicos

A seguir apresentamos as equações de dinâmica e de controle para um pêndulo invertido e para um braço mecânico com duas juntas (*links*). O modelo do pêndulo invertido é usado no capítulo 4 para exemplificar como o arquivo com extensão iv é codificado.

O modelo dinâmico apresentado para ambos os sistemas foi gerado a partir da análise dos sistemas pela formulação Euler-Lagrange. As equações de movimento são descritas em função das coordenadas generalizadas definidas para cada um dos sistemas.

O modelo de controle ou lei de controle é um conjunto de equações que proporciona o controle da movimentação do pêndulo e do braço mecânico. Para o sistema do pêndulo a lei de controle atua de forma a equilibrar o pêndulo na vertical. Para o sistema do braço mecânico a lei de controle atua na movimentação do braço de acordo com os comandos de movimentação. Para ambos os sistemas a lei de controle é do tipo proporcional conforme o que foi apresentado na sessão 3.1.1.

3.3.1 Pêndulo invertido

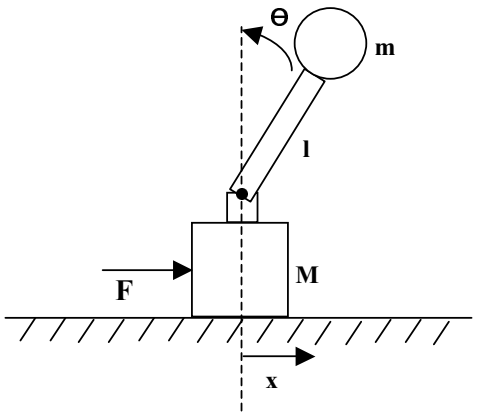


Figura 3-13 - Esquema de um pêndulo invertido

O pêndulo invertido é um sistema não linear clássico que sempre é usado para a análise de controladores. Este sistema é composto por uma base que possui movimento livre na direção x, por uma haste articulada a esta base e por um corpo preso a esta haste, corpo este que se deseja equilibrar em alguma posição na vertical. Os controladores projetados para este sistema geralmente utilizam duas variáveis de controle, o ângulo θ em relação a vertical e a posição **x** da base. Este sistema possui dois graus de liberdade, a saber: a translação **x** sobre o eixo X e a rotação θ no plano XY.

Modelo dinâmico

Com relação à figura 3-13 tem-se variáveis e constantes que compõem o sistema, como apresentado a seguir:

As variáveis são:

θ: ângulo que fornece a posição da haste em relação à vertical (rad);

x: posição da base (m).

As constantes são:

M: massa da base (Kg);

m: massa do pêndulo (Kg);

g: aceleração da gravidade (m/s^2);

l: o comprimento da haste (m);

As equações de dinâmica são:

$$\ddot{x} = \frac{F + gm\cos\theta\,\mathrm{sen}\,\theta - lm\,\mathrm{sen}\,\theta\dot{\theta}^2}{m + M - m\cos^2\theta}$$

$$\ddot{\theta} = \frac{F\cos\theta + gm\,\mathrm{sen}\,\theta + gM\,\mathrm{sen}\,\theta - ml\cos\theta\,\mathrm{sen}\,\theta\dot{\theta}^2}{ml + Ml - ml\cos^2\theta} \qquad \textbf{Equação 3-38}$$

Modelo de controle

O modelo ou lei de controle apresentada a seguir é do tipo proporcional derivativo. Esta lei relaciona a força generalizada com o erro existente entre a posição atual e a posição desejada e com o erro existente entre a derivada da posição atual e a derivada da posição desejada.

$$\tau = Kp\begin{bmatrix} x - x_d \\ \theta - \theta_d \end{bmatrix} + Kd\begin{bmatrix} \dot{x} - \dot{x}_d \\ \dot{\theta} - \dot{\theta} \end{bmatrix} \qquad \textbf{Equação 3-39}$$

3.3.2 Braço mecânico com duas juntas (*links*)

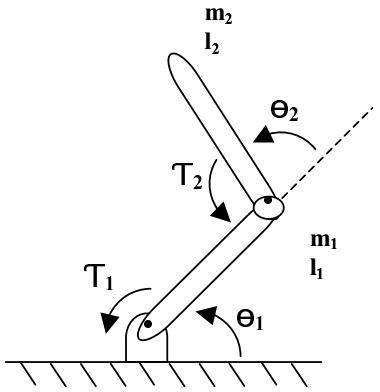


Figura 3-14 - Braço mecânico com duas juntas (*links*)

O braço mecânico com duas juntas, assim como o pêndulo invertido, também é um sistema não linear clássico muito utilizado para a descrição de modelagem dinâmica de sistemas mecânicos. Este sistema possui uma base fixa e duas partes articuladas para formar o braço. Acoplada à base existe um vínculo que une a base à primeira parte do braço. Este vínculo permite que a primeira parte do braço tenha um movimento de rotação no plano XY. Na outra extremidade da primeira parte existe outro vínculo que a

une à segunda parte do braço. Esta segunda parte do braço também possui o movimento de rotação no plano XY. Este sistema possui dois graus de liberdade, a saber: a rotação θ_1 no plano XY da primeira parte do braço e a rotação θ_2 no plano XY da segunda parte do braço.

Modelo dinâmico

Para o sistema apresentado na figura 3-14 tem-se o seguinte conjunto de variáveis e constantes:

As variáveis são:

θ_1: ângulo que fornece a posição da primeira parte do braço em relação à horizontal (rad);

θ_2: ângulo que fornece a posição da segunda parte do braço em relação à horizontal (rad);

As constantes são:

m_1: massa da primeira parte do braço (Kg);

m_2: massa da segunda parte do braço (Kg);

g: aceleração da gravidade (m/s^2);

l_1: o comprimento da primeira parte do braço (m);

l_2: o comprimento da segunda parte do braço (m);

As equações de dinâmica são [CRAIG]:

$$\tau_1 = m_2 l_2^2(\ddot{\theta}_1 + \ddot{\theta}_2) + m_2 l_1 l_2 c_2(2\ddot{\theta}_1 + \ddot{\theta}_2) + (m_1 + m_2) l_1^2 \ddot{\theta}_1 - m_2 l_1 l_2 s_2 \dot{\theta}_2^2 - 2m_2 l_1 l_2 s_2 \dot{\theta}_1 \dot{\theta}_2 + m_2 l_2 g c_{12} + (m_1 + m_2) l_1 g c_1$$

$$\tau_2 = m_2 l_1 l_2 c_2 \ddot{\theta}_1 + m_2 l_1 l_2 s_2 \dot{\theta}_2^2 + m_2 l_2 g c_{12} + m_2 l_2^2(\ddot{\theta}_1 + \ddot{\theta}_2) \qquad \textbf{Equação 3-40} \quad \textbf{(3.50)}$$

onde:

$c_1 = \cos\theta_1$

$s_1 = \operatorname{sen}\theta_1$

$c_{12} = \cos\theta_1 \cos\theta_2 - \operatorname{sen}\theta_1 \operatorname{sen}\theta_2$

$s_{12} = \cos\theta_1 \operatorname{sen}\theta_2 + \operatorname{sen}\theta_1 \cos\theta_2$

As equações 3.40 anteriormente descritas fornecem o torque para cada parte do braço em função da posição, velocidade e aceleração. Pode-se perceber que apesar de simples, pois possui apenas dois graus de liberdade, este sistema apresenta um conjunto de equações demasiadamente complexo, o que se agrava ainda mais quando se trata, por exemplo, de manipuladores com um grau de liberdade ainda maior.

Modelo de controle

O modelo ou lei de controle aqui apresentada é um controle do tipo proporcional derivativo.

$$\tau = K_p \tilde{q} + K_v \dot{\tilde{q}} \qquad \textbf{Equação 3-41}$$

onde:

$\tilde{q} = q_d - q$ é a diferença entre a coordenada generalizada desejada e coordenada generalizada atual.

$\dot{\tilde{q}} = \dot{q}_d - \dot{q}$ é a diferença entre a coordenada generalizada desejada e coordenada generalizada atual.

Em resumo, a lei de controle apresentada na equação 3-41 diz que o torque é proporcional ao erro de posição e proporcional ao erro de velocidade.

Para o sistema de simulação proposto neste trabalho, o usuário deve fornecer a equação de dinâmica na forma apresentada pela equação 3.34. Além desta equação o usuário também deve informar a lei de controle de movimentação do robô.

3.4 Conclusão

Neste capítulo foram apresentados o modelo dinâmico e o modelo de controle do robô, bem como toda a teoria para o estabelecimento destes modelos.

Uma vez estabelecido o modelo de dinâmica, através de uma das formulações apresentadas Newton-Euler ou Lagrange-Euler, a simulação da dinâmica de corpos rígidos pode ser realizada basicamente a partir de duas abordagens: a *on-line* ou implícita e a *off-line* ou explícita. Na sessão 3.2.2 foram apresentadas as vantagens e desvantagens de cada uma destas abordagens e a justificativa para a escolha da abordagem *off-line* para a implementação do sistema de simulação deste trabalho.

Na sessão 3.2.5 foram descritos alguns métodos de integração numérica de equações diferenciais, pois independe da abordagem utilizada para realizar a simulação sempre existe uma etapa de integração numérica das equações diferenciais que descrevem o comportamento dinâmico do robô.

No próximo capítulo é apresentado como e onde o usuário informa o modelo geométrico do robô e as equações de dinâmica e de controle necessárias à simulação.

4 Modelo geométrico

O modelo geométrico, adotado nesta solução, é baseado no modelo geométrico da biblioteca Open Inventor. Primeiramente é mostrado, de maneira simplificada, como a biblioteca funciona e que tipo de funcionalidade ela dispõe aos programadores.

Pode-se citar como vantagens de se utilizar a biblioteca Open Inventor as seguintes características:

- Implementação em linguagem eficiente C++;
- Existência de formato de arquivo para reprodução de cenas e troca de dados com outros sistemas;
- Alto nível de abstração;

Como desvantagens pode-se citar a ausência de editor gráfico com interface 3D que facilite a edição de arquivo padrão da biblioteca Open Inventor. Existem alguns editores que salvam a cena, descrevendo os objetos que a compõe como um conjunto ou malha de pontos (*mesh*), ao invés de descreverem os objetos conforme a sintaxe da gramática do arquivo iv. Na verdade isto não deve ser tratado como um ponto negativo, uma vez que a biblioteca não foi desenvolvida para servir de modelador geométrico e sim para auxiliar, talvez, na implementação de uma aplicação com esta característica.

4.1 Uso do Open Inventor no sistema de simulação

Como já foi dito anteriormente, Open Inventor é uma biblioteca de classes escrita em C++, sob o paradigma orientado a objetos, que permite ao programador criar aplicações gráficas 3D com uma boa utilização dos recursos gráficos de hardware e com baixo esforço de programação [WERNECKE]. Ela é baseada em OpenGL e possui classes de objetos que representam os objetos 3D do mundo real, sendo que estas classes podem ser modificadas ou estendidas de acordo com a necessidade do programador. Desta forma o programador pode trabalhar em um alto nível de abstração, pois ele manipula diretamente um objeto que é equivalente ao objeto real do mundo 3D,

ao invés de trabalhar com um conjunto de *pixels* ou pontos, como é feito em muitas bibliotecas gráficas com baixo nível de abstração.

Para muitas bibliotecas gráficas, o objetivo final é a representação de uma cena 3D através de uma imagem fotorealística. Em Open Inventor o conceito é diferente. O foco é criar um ou vários objetos que representem uma cena 3D e obter como resultado, a apresentação destes objetos em uma tela de computador (renderização), a impressão dos mesmos, o armazenamento em arquivo de suas características para uma utilização posterior e recuperação destas características. A biblioteca Open Inventor mantém armazenadas as características (forma, cor, posição, orientação, textura, etc.) de todos os objetos que pertencem a uma cena. Se a imagem existe somente como um desenho na tela, fica mais complicado para o programador, por exemplo, animar ou mudar a cor de uma parte do desenho. Em OpenGL, por exemplo, existe o conceito de máquina de estados. A imagem é gerada de acordo com as propriedades do estado corrente. Para alterar a imagem devemos alterar as propriedades do estado corrente. Se desejarmos alterar as propriedades como cor, por exemplo, de partes da imagem individualmente, devemos alterar as propriedades do estado corrente exatamente antes do comando de desenho da respectiva parte. Já em Open Inventor cada parte da imagem possui um objeto que a representa e este objeto possui propriedades passíveis de alteração.

O exemplo a seguir mostra um trecho de código para desenhar um cubo em OpenGL e em Open Inventor. O trecho só apresenta o código relativo a definição do cubo. Os outros elementos de definição da cena, tais como fonte de luz e câmera, foram omitidos para simplificar o exemplo.

OpenGL	Open Inventor

glBegin(GL_QUADS);
//Face 0 1 2 3
glVertex3f(-1.0f,-1.0f,-1.0f);
glVertex3f(-1.0f,-1.0f,1.0f);
glVertex3f(-1.0f,1.0f,1.0f);
glVertex3f(-1.0f,1.0f,-1.0f);
 	
//Face 3 2 6 7
glVertex3f(-1.0f,1.0f,-1.0f);
glVertex3f(-1.0f,1.0f,1.0f);
glVertex3f(1.0f,1.0f,1.0f);
glVertex3f(1.0f,1.0f,-1.0f);
 	
//Face 7 6 5 4
glVertex3f(1.0f,1.0f,-1.0f);
glVertex3f(1.0f,1.0f,1.0f);
glVertex3f(1.0f,-1.0f,1.0f);
glVertex3f(1.0f,-1.0f,-1.0f);
 	
//Face 4 5 1 0
glVertex3f(1.0f,-1.0f,-1.0f);
glVertex3f(1.0f,-1.0f,1.0f);
glVertex3f(-1.0f,-1.0f,1.0f);
glVertex3f(-1.0f,-1.0f,-1.0f);
 	
//Face 2 1 5 6
glVertex3f(-1.0f,1.0f,1.0f);
glVertex3f(-1.0f,-1.0f,1.0f);
glVertex3f(1.0f,-1.0f,1.0f);
glVertex3f(1.0f,1.0f,1.0f);
 	
//Face 3 7 4 0
glVertex3f(-1.0f,1.0f,-1.0f);
glVertex3f(1.0f,1.0f,-1.0f);
glVertex3f(1.0f,-1.0f,-1.0f);
glVertex3f(1.0f,-1.0f,-1.0f);
glEnd();
	Usando o arquivo iv para definir o cubo:
	Shape {
	Cube {
	width 2
	height 2
	deph 2
	}
	}
	
	Usando a classe que representa o cubo:
	SoSeparator * raiz = new SoSeparator;
	SoCube * cubo = new SoCube(2);
	
	raiz->ref();
	raiz->addChild(cube);

Tabela 4-1- Código para desenhar um cubo em OpenGL e em Open Inventor

O exemplo da tabela 4-1 mostra que no código em OpenGL o cubo é definido por seis faces e cada uma destas por quatro de vértices (pontos em 3D), já no código em Open Inventor o cubo é definido pela instanciação de um objeto da classe SoCube.

Como a biblioteca Open Inventor armazena informações sobre os objetos tais quais as dos objetos no mundo real e não somente um conjunto de pontos a serem desenhados na tela, outras operações além da renderização podem ser realizadas sobre os objetos. Em uma cena, os objetos podem ser: selecionados, destacados, impressos, lidos ou escritos em arquivos, animados, modificados, ou qualquer outra operação permitida. Em Open Inventor os objetos são manipulados como entidades discretas.

A figura 4-1 mostra a arquitetura básica da biblioteca Open Inventor, através da organização das classes em grandes blocos interelacionados.

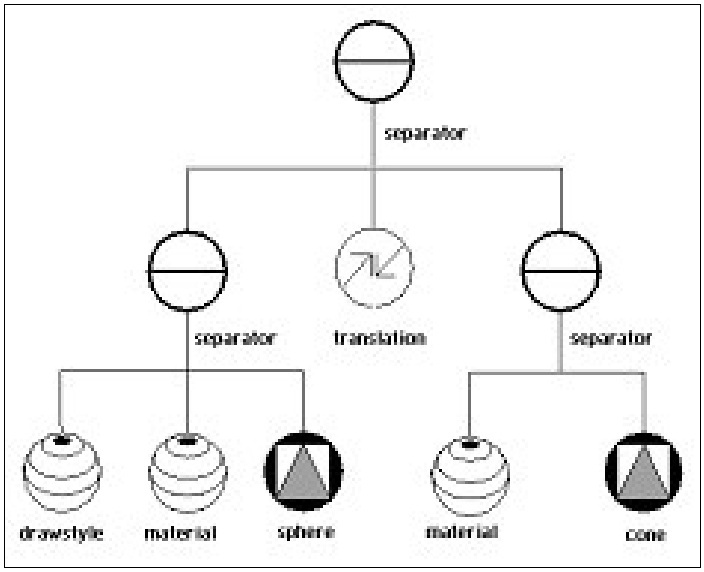


Figura 4-1- Arquitetura da biblioteca Open Inventor [WERNECKE]

As classes de objetos desta biblioteca estão divididas em quatro grandes grupos de acordo com suas características funcionais. De acordo com a figura 4-1 temos:

- **Banco de dados da cena *(Scene Database)***: São classes de objetos que representam os objetos 3D do mundo real e suas características físicas. Por exemplo, suponha que se deseje criar uma cena com uma bola azul. Este objeto 3D do mundo poderia ser representado por um objeto geométrico esfera e um material de cor azul e opaco. A forma, o tamanho, o material, a cor, a posição, a orientação e outras propriedades são características de

objetos do mundo 3D que são representadas por estas classes. Uma cena em Open Inventor é composta de nós ligados em uma certa ordem. A este conjunto de nós ordenados é dado o nome de grafo da cena, na figura 4-2 podemos ver um exemplo de grafo de cena.

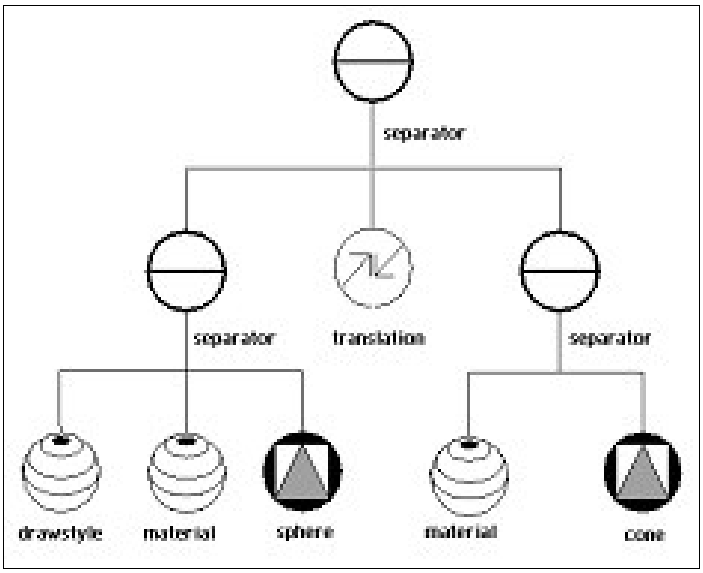


Figura 4-2 - Exemplo de grafo de cena

Cada nó do grafo da cena representa uma parte da informação gráfica da cena. Os nós podem representar forma, cor, material, luz, câmera, transparência, textura, etc. Existe uma classe para representar cada tipo de nó que compõe o grafo da cena. Os ícones, com os quais o grafo da cena da figura 4-2 foi representado, fazem parte de um conjunto de ícones padrão para a representação de grafos da cena, como é mostrado na figura 4-3 a seguir. As classes que pertencem a este grupo tem o nome de banco de dados da cena, porque todas as informações contidas em uma cena são armazenadas em objetos de acordo com o tipo da informação. Por exemplo, se na cena existe uma fonte de luz e um objeto geométrico cubo, um objeto da classe luz e um objeto da classe forma geométrica são instanciados para armazenar todos os atributos de cada um destes objetos. Quando se usa a expressão banco de dados da cena, é uma referência aos objetos instanciados que representam as informações contidas em uma cena.

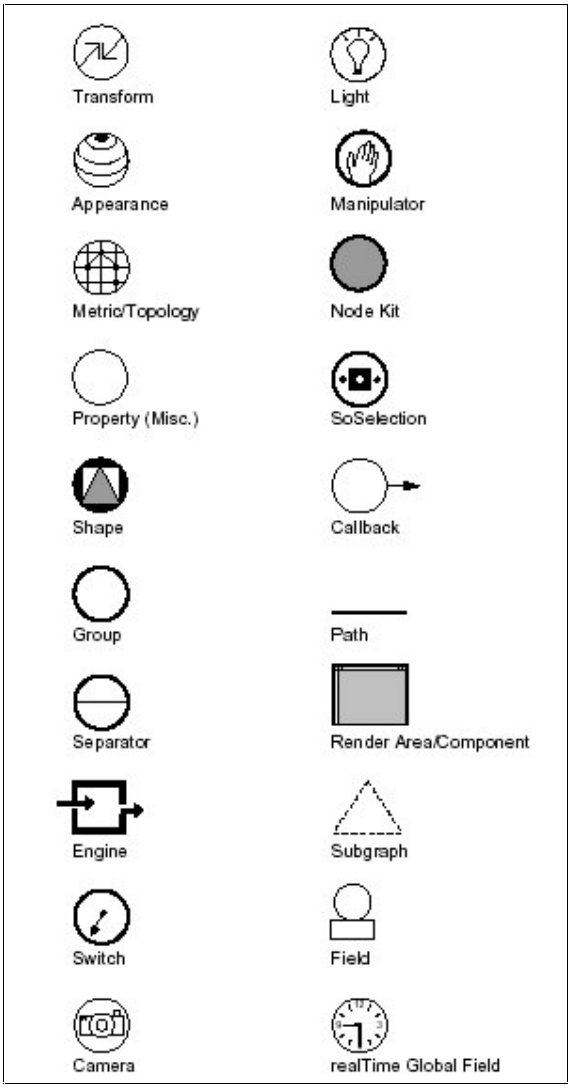


Figura 4-3 - Ícones que representam os nós de um grafo de cena em Open Inventor

- **Kits de nós *(Node Kits)***: São classe de objetos que representam conjuntos de nós, com uma determinada estruturação. Esta estruturação facilita a construção de cenas com maior complexidade.
- **Manipuladores *(Manipulators)***: São classes de objetos que provêem diversas formas de seleção dos objetos 3D de uma cena. Este conjunto de classes possui objetos que tratam os eventos gerados pelos dispositivos apontadores (*mouse*, *joystick*, *trackball*, e outros). Com objetos destas classes é possível, por exemplo, alterar a cor de um objeto ao clicar com o *mouse* sobre o mesmo. O evento "clicar com o mouse" é capturado e traduzido em uma alteração no banco de dados da cena para mudar a cor do

objeto que recebeu o clique. Desta forma, é possível construir aplicações que permitam uma interação do usuário com a cena, o usuário pode selecionar, destacar, ou até mesmo buscar um objeto da cena através destas classes. Estas classes de objetos provêem uma forma fácil de manipulação de objetos 3D da cena, pois elas simulam: o *mouse*, o *trackball*, o *joystick* e as caixas envoltórias. A figura 4-4 mostra uma cena com objetos e os vários tipos de manipuladores existentes.

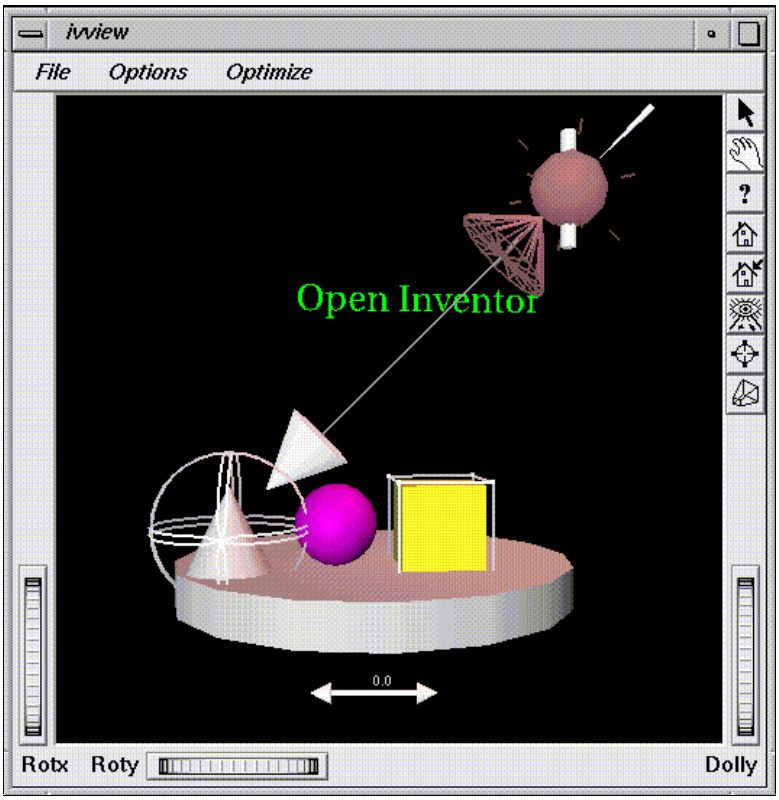


Figura 4-4 - Exemplos de manipuladores (selecionadores) de objetos de uma cena em Open Inventor

- **Componentes *(Open Inventor Component Library)***: São classes de objetos que auxiliam a construção de uma aplicação com uma interface gráfica. As classes que fazem parte deste grupo são para a construção de aplicativos sobre o sistema de janelas e eventos X Windows, por isso estas classes não fazem parte do núcleo principal da biblioteca Open Inventor, como pode ser visto na figura 4-1. Existem versões deste conjunto de componentes para cada tipo de interface: uma versão para X Windows, uma versão para

Windows e uma versão escrita em Qt que é multiplataforma. Neste conjunto de componentes existem objetos para renderizar as cenas, para traduzir eventos e mensagens do X Windows para eventos e mensagens do Open Inventor, para editar propriedades tais como, luz, cor e material, assim como objetos para visualizar e editar cenas escritas no formato do Open Inventor. O objeto janela que contém a área de renderização está preparado para receber eventos do sistema X Windows e traduzi-los em eventos do Open Inventor como mostra a figura 4-5 a seguir. Este conjunto de classes forma um *framework* para a construção de aplicações com uma interface gráfica e são chamados de componentes Xt (X *translator*). Podemos estendê-los, ou criar novos componentes deste tipo para utilizarmos em nossas aplicações.

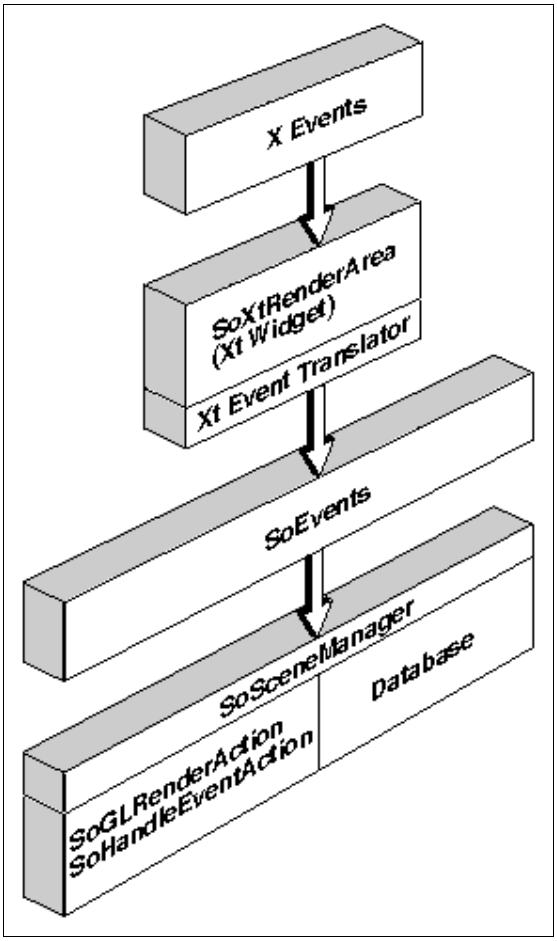


Figura 4-5 - Processamento de eventos na biblioteca Open Inventor [WERNECKE]

As figuras 4-6, 4-7 e 4-8 a seguir mostram a hierarquia de classes da biblioteca Open Inventor de forma condensada, as classes ancetrais estão à esquerda e as classes

descendentes estão à direita. Um diagrama de classes completo pode ser consultado no Apêndice B.

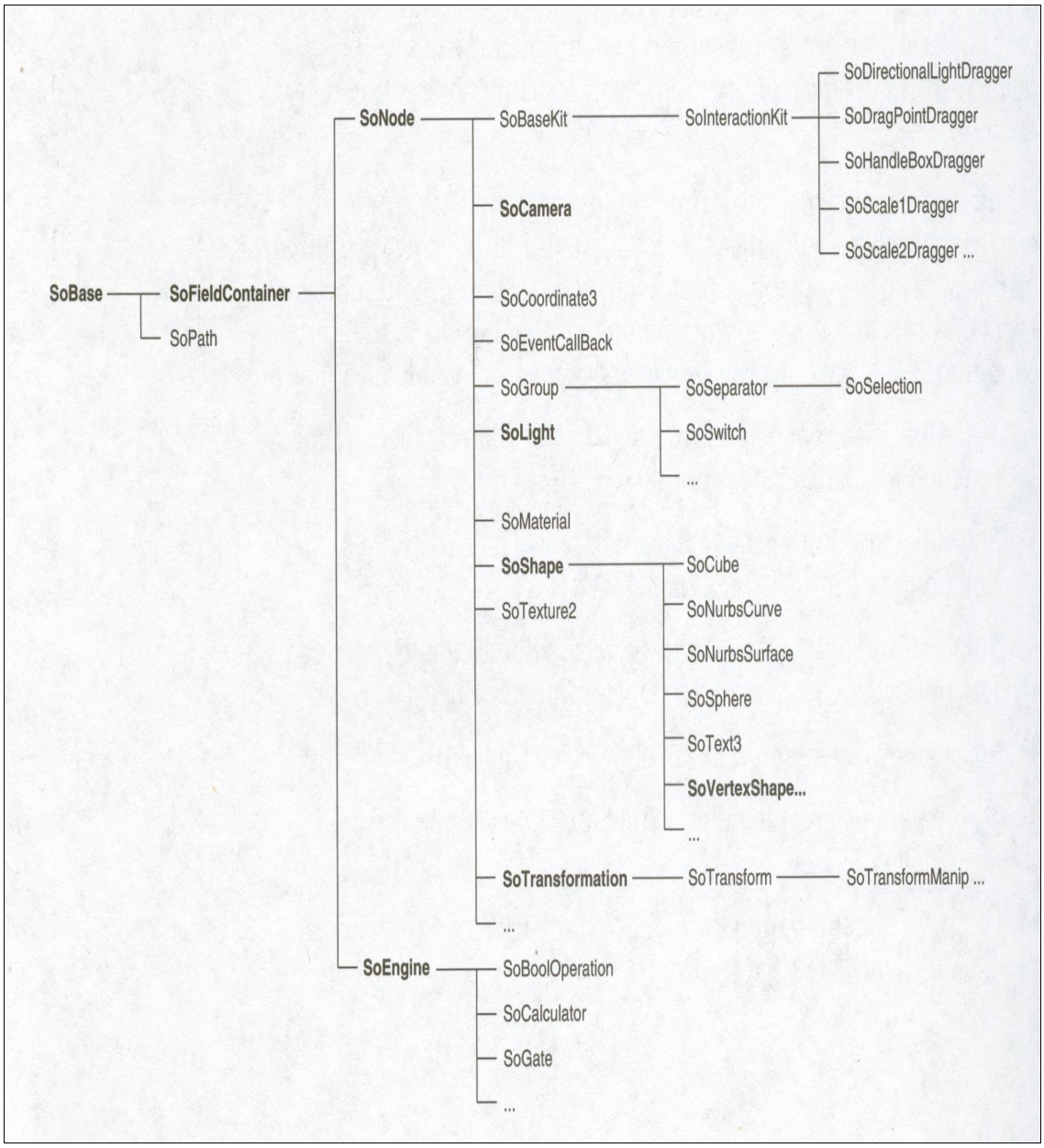


Figura 4-6 - Hierarquia de classes da biblioteca Open Inventor (parte 1 de 3) [WERNECKE]

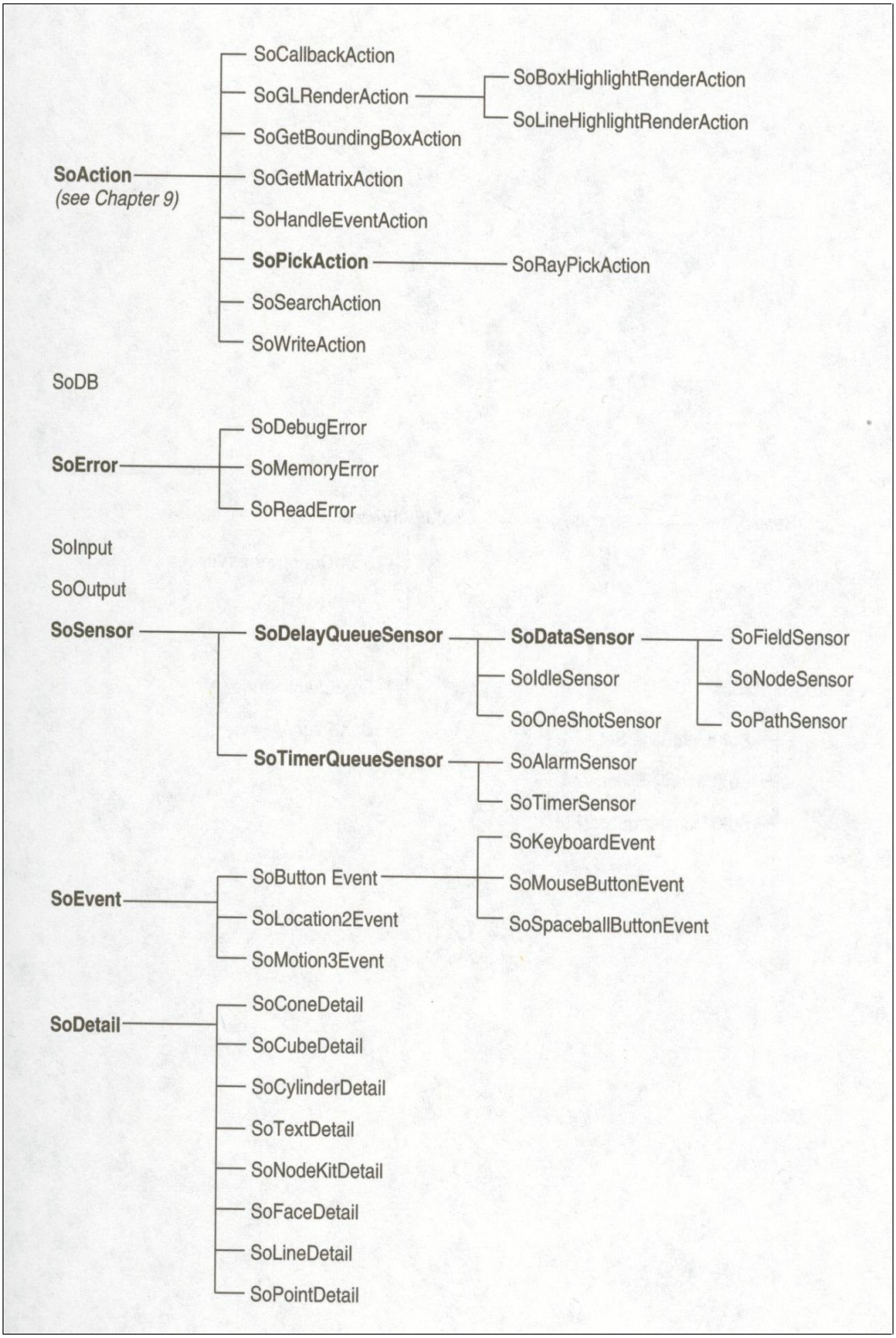


Figura 4-7 - Hierarquia de classes da biblioteca Open Inventor (parte 2 de 3) [WERNECKE]

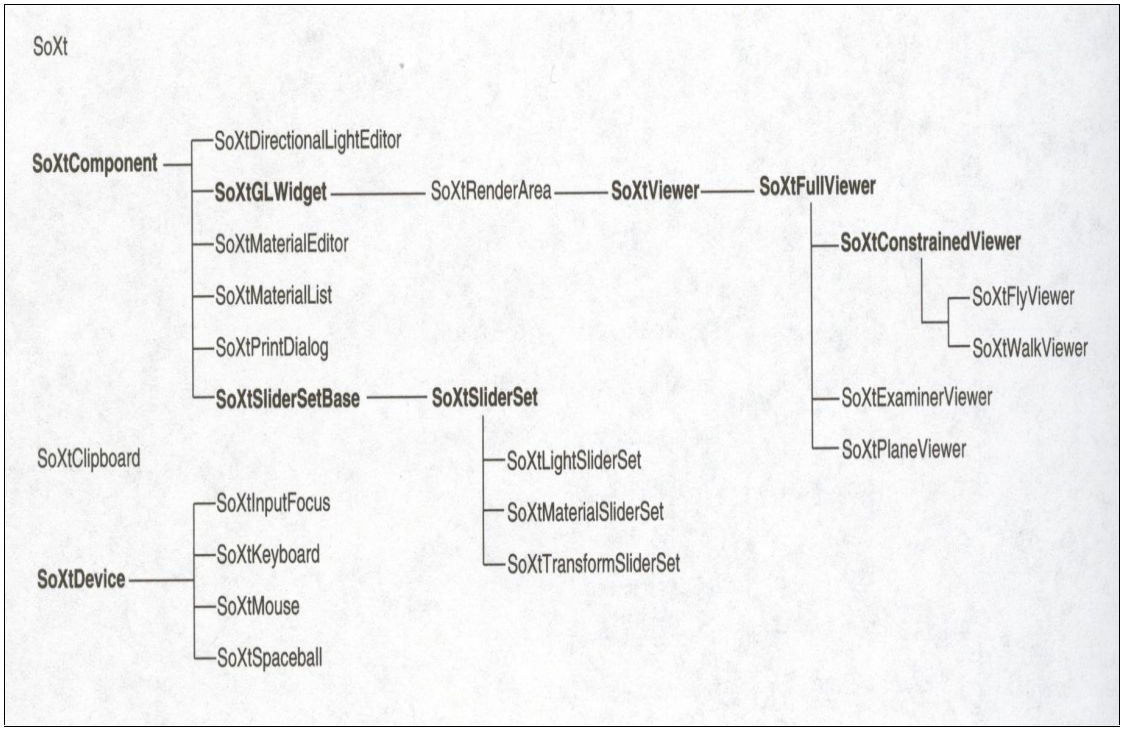


Figura 4-8 - Hierarquia de classes da biblioteca Open Inventor (parte 3 de 3) [WERNECKE]

As classes SoBase e suas sub-classes representam os objetos 3D do mundo real. As classes SoAction, SoSensor, SoEvent e suas sub-classes representam o comportamento dos objetos 3D do mundo real. As classes SoXt e suas sub-classes representam os componentes Xt. Existem outras classes auxiliares tais como SoError para tratamento de erro, e SoInput e SoOutput que fazem a leitura e escrita de uma cena.

Uma das grandes vantagens de se utilizar esta biblioteca é facilidade de extensão de sua funcionalidade através da criação de novos objetos por herança, ou seja, aplicação direta dos conceitos de programação orientada a objetos e através de classes que tratam eventos e mensagens. Estas classes possuem métodos para chamar funções definidas pelo usuário (*callback functions*), quando um determinado evento ou mensagem ocorre. Por exemplo, existe uma classe para tratar o evento de seleção de objetos de uma cena com o *mouse*. Esta classe possui um método para o qual é passada, como parâmetro, uma função do usuário para que esta seja executada quanto o evento de seleção vier a ocorrer. Desta forma a extensão de funcionalidade fica facilitada tanto pelo mecanismo de herança como pelo mecanismo de *callback functions*.

4.2 Um programa como exemplo de utilização de Open Inventor

Para ilustrar é apresentado um exemplo simples de uma aplicação que cria uma esfera azul e a apresenta em uma janela. Neste exemplo, a cena é renderizada em uma janela que faz parte dos Componentes da biblioteca Open Inventor, ou seja, um componente Xt.

Para realizar tal tarefa constrói-se o grafo da cena que contém os seguintes nós: um nó de câmera, um nó de luz, um nó de material e um nó de objeto geométrico (a esfera). O grafo da cena pode ser construído no próprio código em C++, através da instanciação de objetos das classes que representam o nós citados, ou através de um arquivo de descrição do grafo da cena escrito em ASCII de acordo com o formato Open Inventor. Os trechos de código apresentados na figura 4-9 e na figura 4-10 mostram as duas possibilidades respectivamente. A figura 4-12 mostra o grafo da cena para estes exemplos e a figura 4-13 mostra a cena renderizada e apresentada na janela que é um componente Xt.

Exemplo 1 - Grafo da cena construído em C++

```
#include <Inventor/Xt/SoXt.h>
#include <Inventor/Xt/SoXtRenderArea.h>
#include <Inventor/nodes/SoSphere.h>
#include <Inventor/nodes/SoDirectionalLight.h>
#include <Inventor/nodes/SoMaterial.h>
#include <Inventor/nodes/SoPerspectiveCamera.h>
#include <Inventor/nodes/SoSeparator.h>

main() {
       //Inicializa os objetos do Inventor e
       //retorna a janela principal da aplicação
       Widget janela = SoXt::init();
       if (janela == NULL) exit (1);

       //Criação dos objetos que fazem parte do grafo da cena
       SoSeparator * raiz               = new SoSeparator;
       SoPerspectiveCamera * camera     = new SoPerspectiveCamera;
       SoDirectionalLight * luz         = new SoDirectionalLight;
       SoMaterial * material            = new SoMaterial;
       SoSphere * esfera                = new SoShere;

       //Alteração de algumas propriedades dos objetos
       //Define a cor do material (vermelho);
       material->diffuseColor.setValue(1.0,0.0,0.0);

       //Estruturação do grafo da cena por adição da câmera, da luz, do
       //material e da esfera abaixo do nó separador
       raiz->ref();
       raiz->addChild(camera);
       raiz->addChild(luz);
       raiz->addChild(material);
       raiz->addChild(esfera);

       //Cria a área de renderização na qual a cena é apresentada
       SoXtRenderArea * areaRender = new SoXtRenderArea(janela);

       //Prepara a câmera para mostrar toda a área de renderização
       camera->viewAll(raiz, areaRender->getViewportRegion());

       //Coloca a cena na área de renderização, troca o título e
       //apresenta a cena
       areaRender->setSceneGraph(raiz);
       areaRender->setTitle("Olá Esfera");
       areaRender->show();

       SoXt::show(janela); //Apresenta janela principal
       SoXt::mainLoop();   //Ciclo principal do Inventor
}
```

Figura 4-9 - Exemplo 1

Exemplo 2 - Grafo da cena lido de um arquivo ASCII no formato Open Inventor

```
#include <Inventor/Xt/SoXt.h>
#include <Inventor/Xt/SoXtRenderArea.h>
#include <Inventor/nodes/SoSeparator.h>

main() {
	//Inicializa os objetos do Inventor e
	//retorna a janela principal da aplicação
	Widget janela = SoXt::init();
	if (janela == NULL) exit (1);

	//Inicializa o banco de dados do Open Inventor
	SoDB::init();

	//Manipulação do arquivo que contém a cena
	SoInput grafoArq;
	//Abre o arquivo
	if ( !(grafoArq.openFile("c:\mmm\esfera.iv")) ){
	cout << "\nNão foi possível abrir o arquivo
		de definição do grafo da cena" << endl;
	}
	//Le o grafo da cena contida no arquivo
	SoSeparator * raiz = SoDB::readAll(&grafoArq);
	if (raiz == NULL) {
		cout << "\nNão foi possível ler o arquivo de definição da cena"
		<< endl;
	}
	//Fecha o arquivo
	grafoArq.closeFile();

	raiz->ref();

	//Cria a área de renderização na qual a cena é apresentada
	SoXtRenderArea * areaRender = new SoXtRenderArea(janela);

	//Coloca a cena na área de renderização, troca o título e
	//apresenta a cena
	areaRender->setSceneGraph(raiz);
	areaRender->setTitle("Olá Esfera");
	areaRender->show();

	SoXt::show(janela); //Apresenta janela principal
	SoXt::mainLoop();   //Ciclo principal do Inventor
}
```

Figura 4-10- Exemplo 2

O conteúdo do arquivo “esfera.iv” é o mostrado na figura 4-11.

```
#Inventor v2.0 ascii
Separator {
	PerspectiveCamera { }
	DirectionalLight { }
	Material {
		DiffuseColor 1.0 0.0 0.0
	}
	Shape {
		Sphere {}
	}
}
```

Figura 4-11 - Conteúdo do arquivo "esfera.iv"

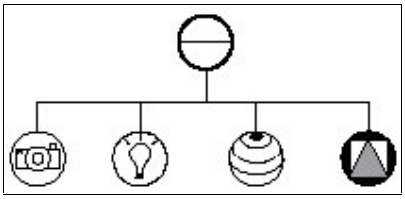


Figura 4-12 - Grafo da cena para os exemplos 1 e 2

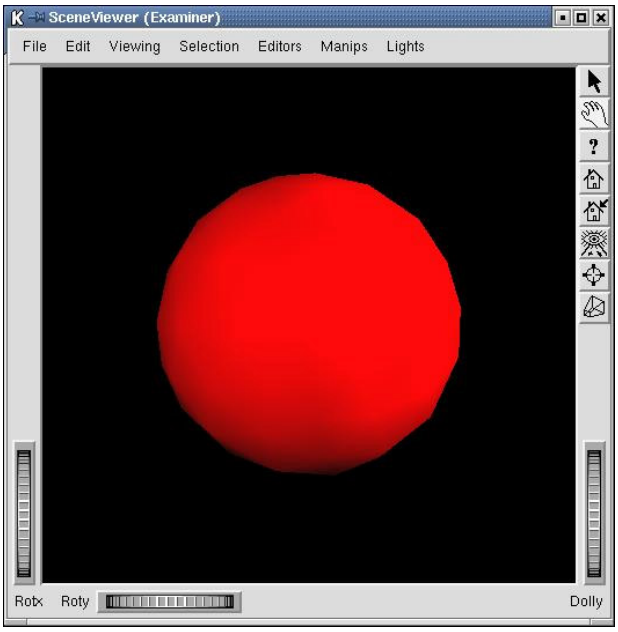


Figura 4-13 - Cena renderizada da esfera vermelha

A partir destas duas possibilidades de se apresentar uma cena, decidiu-se usar o arquivo ASCII no formato Open Inventor para servir de entrada para o nosso sistema de simulação. Neste arquivo o usuário informa quais são as partes que compõem o robô a ser simulado seguindo o padrão Open Inventor de descrição de grafo da cena. A definição das partes que compões o robô pode ser feita utilizando objetos geométricos simples, tais como cubo, prisma, esfera, etc, ou pode ser feita utilizando objetos geométricos mais complexos descritos através de conjuntos de pontos.

4.3 Extensão do arquivo IV

O arquivo iv provê para a simulação, o modelo geométrico do robô. Porém, além do modelo geométrico, o usuário precisa informar as equações de controle e de dinâmica do robô e obrigatoriamente precisa relacionar as partes do modelo geométrico do robô com as variáveis das equações de controle e de dinâmica. Para isso foi necessário estender o arquivo iv introduzindo estas informações através de nós não renderizáveis.

Para se explicar como foi feita a extensão do arquivo padrão da biblioteca Open Inventor é preciso, inicialmente, entender este formato padrão. A sintaxe para escrita do arquivo ASCII do Open Inventor por ser resumida em:

- Linhas de comentário: linha iniciadas pelo o símbolo #.

Exemplo:

```
#linha de comentário
```

- Linha de cabeçalho: deve ser a primeira linha do arquivo e serve para indicar a versão do arquivo e se sua codificação está em ASCII ou em binário.

Exemplo:

```
#Inventor v2.0 ascii
```

ou

```
#Inventor v2.0 binary
```

- Linhas para definir os nós que compõem o grafo da cena.

Exemplo:

```
Transform {
            Tranlation 0 -4 0.2
}
```

A gramática a ser seguida para a escrita do arquivo é bem simples:

```
file-asc := header body
body := node body | empty
node := node-name { body | fields }
fields := field fields | empty
field := field-name field-value
```

Figura 4-14 - Gramática de definição do arquivo iv

De acordo com a gramática da figura 4-14 tem-se:

- O nome do nó
- Um abre chave ({)
- Os campos que pertencem ao nó ou sub-nó
- Um fecha chave (})

Exemplo:

```
DrawStyle {
        style         LINES
        lineWidth     3
        linePattern   255
}
```

No exemplo anterior o nome do nó é `DrawStyle`. Os campos deste nó são: `style`, `lineWidth` e `linePattern`; cada um com seus respectivos valores.

Cada tipo de nó possui seu conjunto específico de campos. No manual de referência do Open Inventor [OPEN INVENTOR] pode ser encontrada a descrição do formato do arquivo para cada tipo de nó.

Ao definirmos um nó é possível dar-lhe também um nome.

Exemplo:

```
DEF Cubo_01 Cube {
        width         2.0
        height        2.0
        depth         2.0
}
```

Onde: `Cubo_01` é o nome dado ao nó cujo tipo é um objeto geométrico `Cube` e cujas propriedades são `width`, `height` e `depth`.

Fez-se então uma extensão deste arquivo para que o mesmo contivesse outras informações além da geometria do robô. Esta extensão foi realizada mantendo-se a compatibilidade com o formato padrão do arquivo. Nenhuma nova funcionalidade foi adicionada à biblioteca do Open Inventor para a interpretação desta extensão. Utilizou-se o artifício de introduzir nós no grafo da cena que não são renderizáveis e, nestes nós, incluiu-se as informações sobre a dinâmica e controle do robô. Este tipo de nó, não renderizável, é um nó que não descende de nenhum nó que possui propriedades visuais tais como cor, forma, material, posição, tamanho, etc.

O arquivo de descrição do robô é fornecido como entrada para o sistema de simulação e após a leitura deste arquivo os nós que possuem propriedades visuais, são

renderizados e a cena é apresentada com o robô em sua condição inicial de posição e velocidade; os nós, que não possuem propriedades visuais, são lidos e as informações que eles contém são utilizadas para extrair as equações de dinâmica e as leis de controle que são utilizadas para calcular o novo estado do robô a partir de um comando de movimentação.

O objetivo desta extensão é fazer com que o modelo geométrico, as equações de dinâmica e de controle e as relações entre o modelo geométrico e estas equações sejam descritos em um único lugar: o arquivo de descrição do robô.

Como é possível estender os arquivos do Open Inventor no formato ASCII, criando nós não padrão do Open Inventor, decidiu-se então incluir no arquivo de descrição geométrica as coordenadas generalizadas associadas a cada parte do robô, as características dinâmicas (equações, variáveis e constantes) e as leis de controle do robô que será simulado. O arquivo entendido ficou então com a seguinte estrutura:

```
#Inventor V2.0 ascii
Separator {

  DEF parte0 Group {
    DEF parte0geometria Group {
    #propriedades geométricas da parte 0
    }
    DEF parte0dinamica Group {
    #propriedades dinâmicas da parte 0
    }
  }

  DEF parte1 Group {
    DEF parte1geometria Group {
    #propriedades geométricas da parte 1
    }
    DEF parte1dinamica Group {
    #propriedades dinâmicas da parte 1
    }
  }
  ...

  DEF parteN Group {
    DEF parteNgeometria Group {
    #propriedades geométricas da parte N
    }
    DEF parteNdinamica Group {
    #propriedades dinâmicas da parte N
    }
  }

  DEF globais Group {
    DEF g0 G0 {
    #definição da constante global 0
    }
    DEF g1 G1 {
    #definição da constante global 1
    }
    ...
    DEF gN GN {
    #definição da constante global N
    }
  }
```

Definição das propriedades:

1) Geométricas
- forma
- cor
- posição
- orientação

2) Dinâmicas
- Coordenada generalizada associada à parte
- Derivada da coordenada generalizada associada à parte

Definição de constantes globais

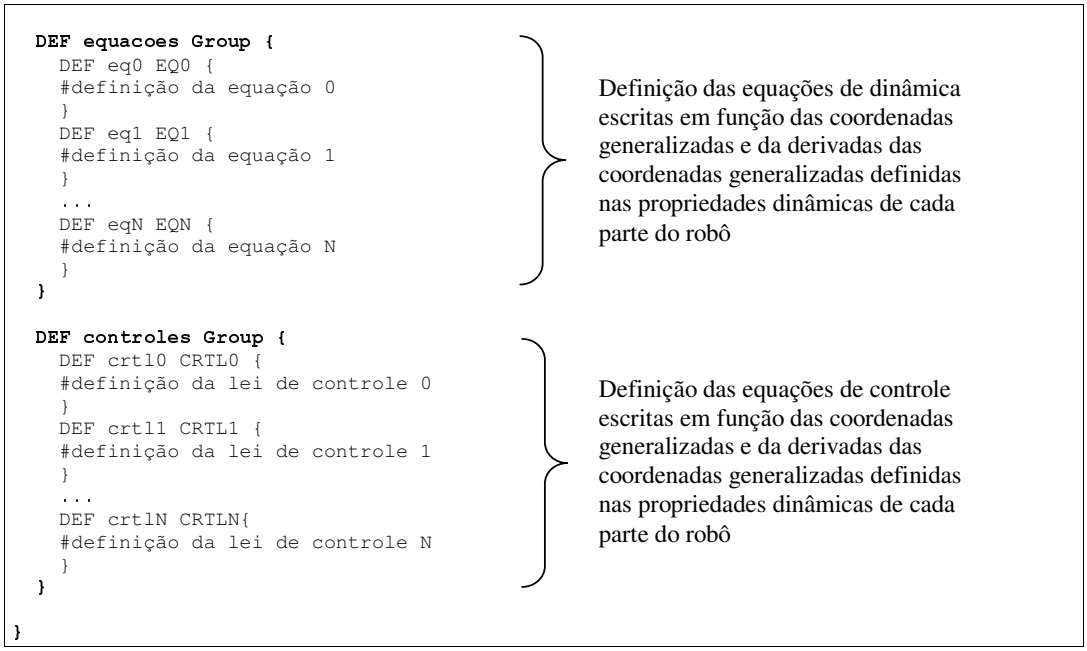
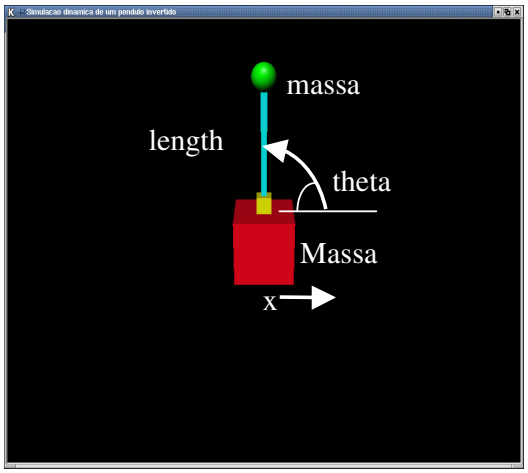


Figura 4-15 - Estrutura do arquivo iv estendido

A estrutura apresentada na figura 4-15 mostra que o arquivo iv estendido é formado inicialmente grupos que descrevem as propriedades geométricas e a dinâmicas de cada parte do robô, seguido por um grupo que descreve as constantes globais, por um grupo que descreve as equações de dinâmica e por último um grupo que descreve as equações de controle. Para um robô composto de M partes, o arquivo tem um total de $N+3$ grupos, onde $N = M-1$ pois a numeração das partes é iniciada em zero.

Para exemplificar como ficaria o arquivo iv com as extensões propostas e como a simulação ocorre, suponha o sistema a seguir.

A figura 4-16 a seguir mostra a imagem renderizada de um pêndulo invertido e suas equações de dinâmica.



$\theta \rightarrow$ *deslocamento angular em z*
$x \rightarrow$ *deslocamento linear em x*
$M \rightarrow$ *massa da base*
$m \rightarrow$ *massa do corpo equilibrado*
$l \rightarrow$ *comprimento do pêndulo*
$g \rightarrow$ *força da gravidade*
$\tau \rightarrow$ *força ou torque*

$$\ddot{\theta} = \frac{((M+m)*g*\theta - \tau)}{M*l}$$

$$\ddot{x} = \frac{(\tau - m*g*\theta)}{M}$$

Figura 4-16 - Pêndulo invertido e suas equações de dinâmica

Na simulação existe uma classe responsável por calcular o novo estado dinâmico do pêndulo, ou seja, posições, velocidades e acelerações das partes que compõem o pêndulo. Para que o módulo de animação saiba quais partes do pêndulo tiveram suas posições alteradas, é preciso relacionar as variáveis das equações de dinâmica com as respectivas partes do pêndulo sobre as quais estas variáveis têm efeito. Na figura 4-16, o deslocamento angular em Z está associado à haste que equilibra o corpo e o deslocamento linear em x está associado à base do pêndulo. A cada cálculo de um novo estado dinâmico, as variáveis que estão relacionadas com o modelo geométrico são pesquisadas afim de que se obtenha seu novo valor e este novo valor seja passado para o módulo de animação para que este redesenhe o pêndulo em sua nova posição.

O pêndulo invertido da figura 4-17 será construído a partir de formas geométricas básicas tais como cubo e esfera que estão descritas no arquivo iv. Neste arquivo também estão incluídas as extensões que descrevem as coordenadas generalizadas associadas a cada parte do robô, as características dinâmicas (equações, variáveis e constantes) e as leis de controle do robô.

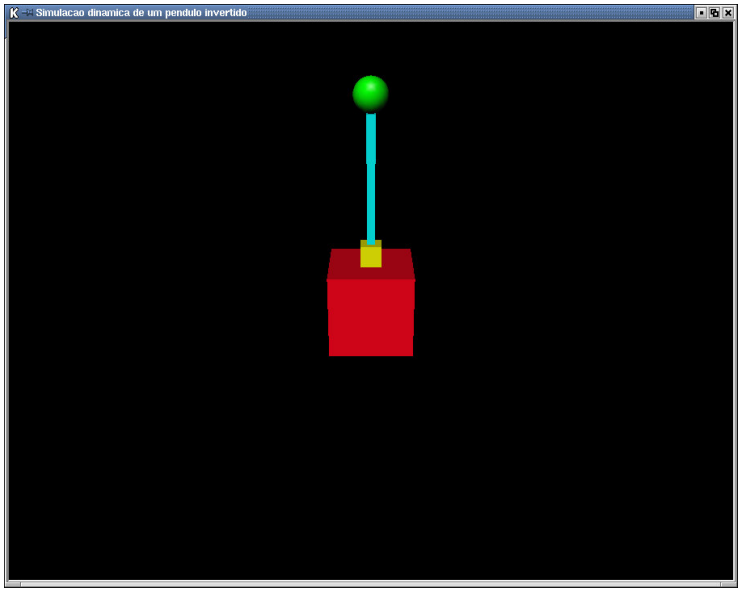


Figura 4-17 - Pêndulo invertido

Conteúdo do arquivo "pendulo.iv" que representa a cena da figura 4.17 é apresentado a seguir. Os nós que fazem parte da extensão, nós não visuais, estão destacados dos demais em negrito. Este arquivo foi criado baseado no modelo simplificado de um pêndulo como mostra a figura 4-18.

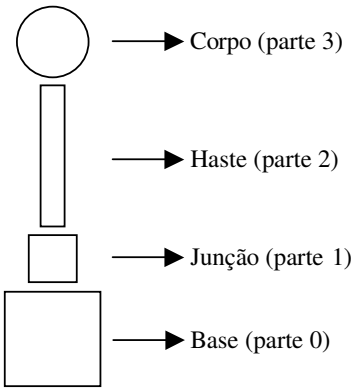


Figura 4-18 - Sub-partes do pêndulo invertido

Figura 4-19 – (a) Conteúdo do arquivo "pendulo.iv" que descreve o sistema proposto na figura 4-16

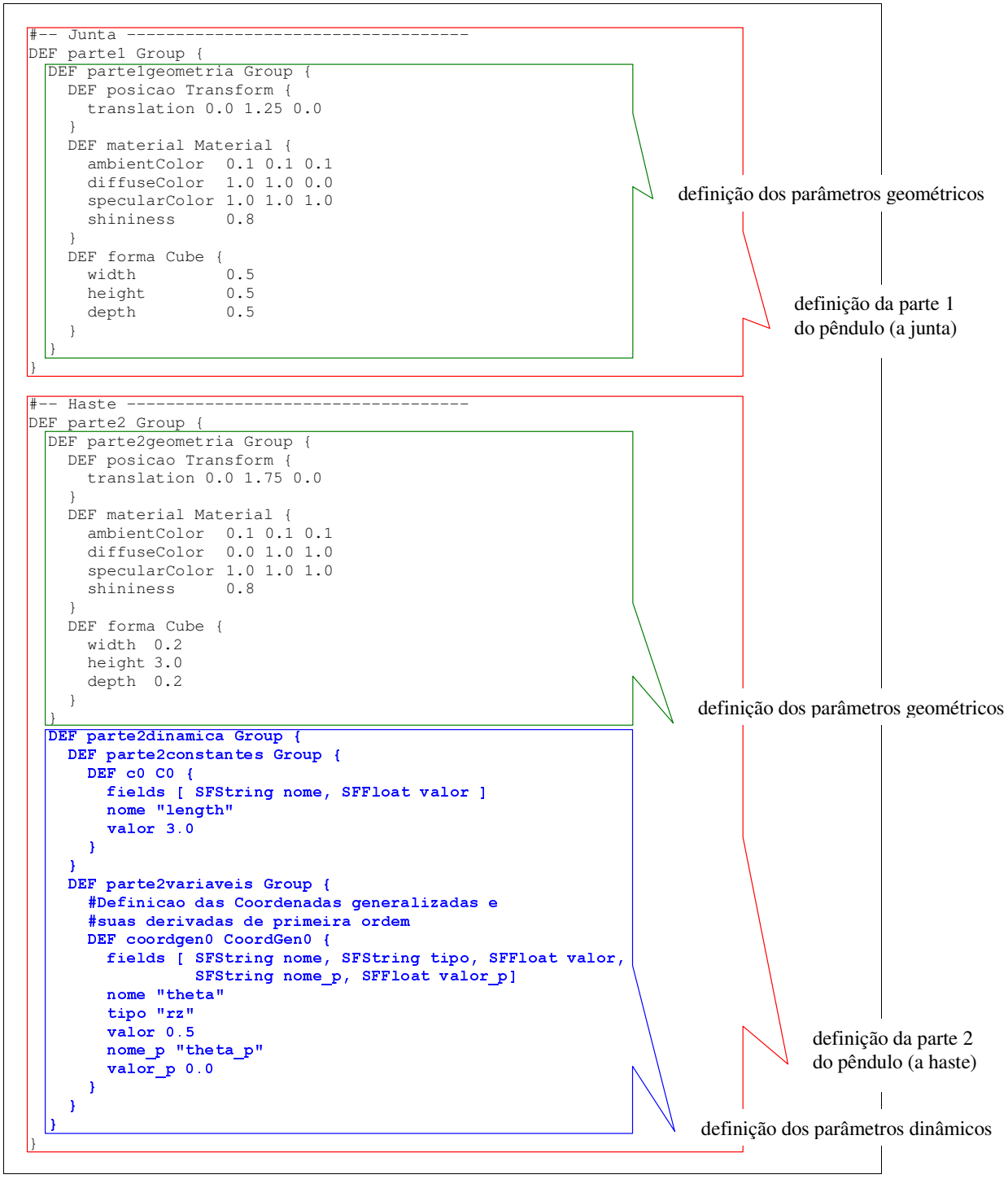


Figura 4-19 – (b) Conteúdo do arquivo "pendulo.iv" que descreve o sistema proposto na figura 4-16

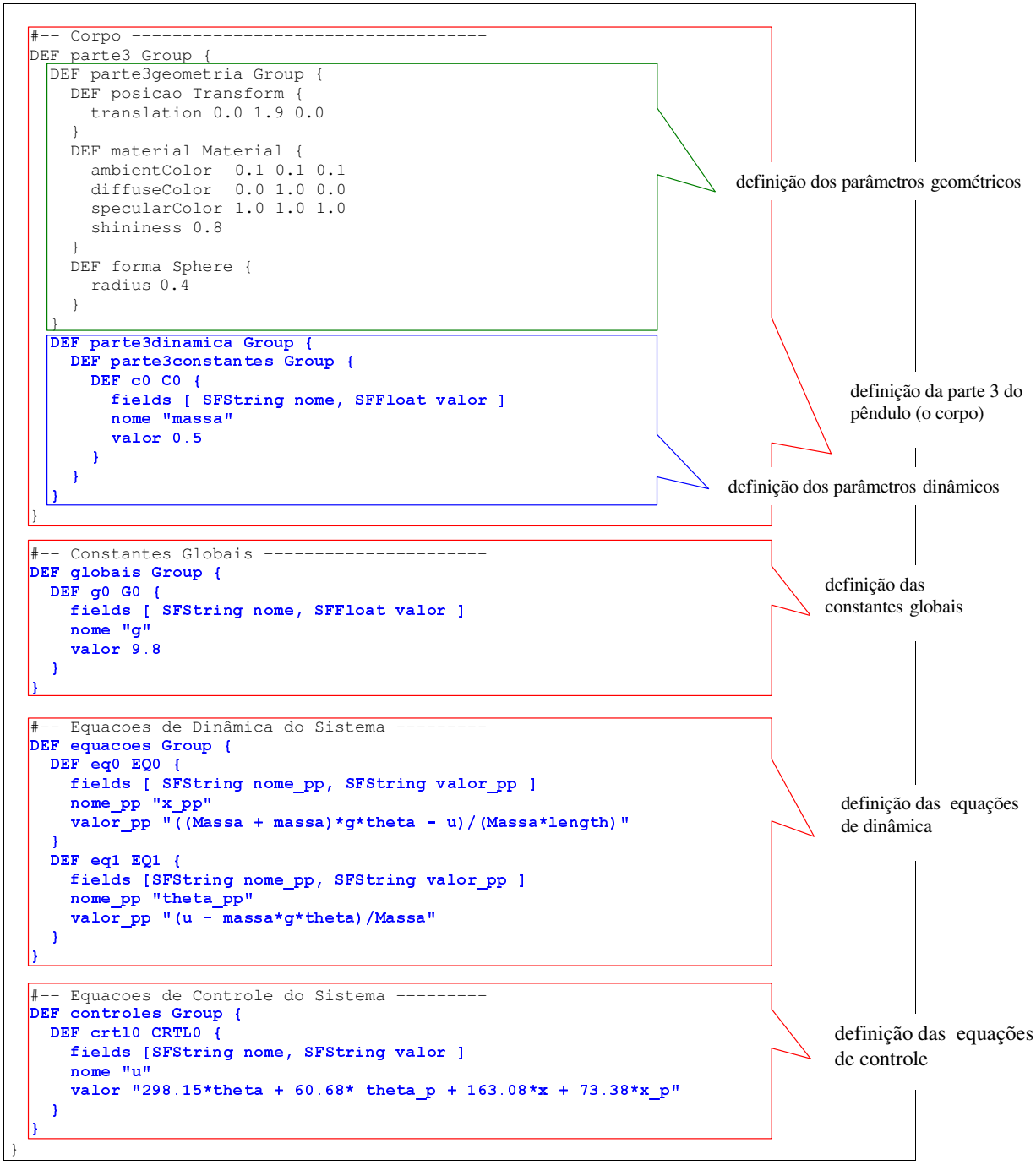


Figura 4-19 – (c) Conteúdo do arquivo "pendulo.iv" que descreve o sistema proposto na figura 4-16

Para a edição deste arquivo pode ser usado qualquer editor de arquivo texto. Este arquivo também pode ser gerado por um editor específico a ser implementado em um trabalho futuro.

Os modelos geométricos geralmente possuem uma hierarquia de construção *bottom-up* [FOLEY]. Componentes básicos são usados como blocos de montar para criar objetos geométricos mais complexos e estes, por sua vez, também podem ser usados como partes para compor objetos com um nível de complexidade ainda maior. Quando não existe a repetição de um determinado componente básico em toda a hierarquia, o que não é muito comum, a mesma pode ser representada por uma estrutura em árvore. Quando um componente básico se repete em diversos pontos da hierarquia, a mesma, é representada por uma estrutura em grafo acíclico ou *directed acyclic graph* (DAG) [WALSH]. A figura 4-20 mostra um robô humanóide em perspectiva, sendo que o grafo (a) é a representação da estrutura do robô em DAG e o grafo (b) é a representação da estrutura do robô em árvore. A representação em grafo acíclico evita a repetição de partes da estrutura que são idênticas, como por exemplo, os braços esquerdo e direito do robô. O arquivo padrão da biblioteca Open Inventor também aceita que um cena seja descrita com um grafo do tipo DAG, define-se o objeto uma única vez e depois ele é referenciado toda vez que se repetir no grafo.

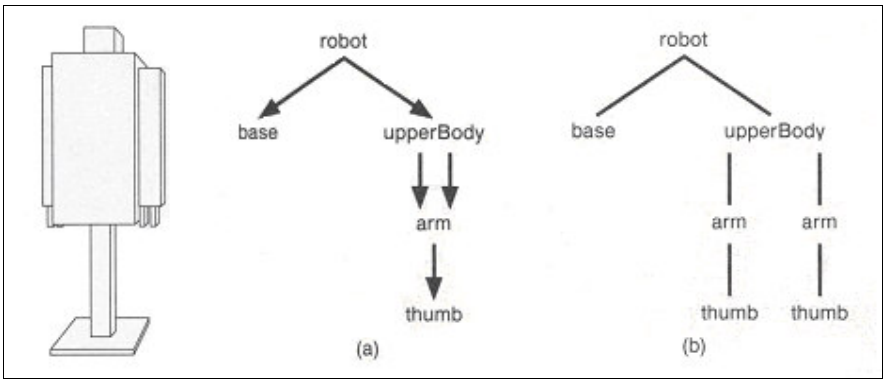


Figura 4-20 - Hierarquia dos componentes de um robô. (a) DAG. (b) árvore [FOLEY]

Existe uma certa hierarquia a ser seguida para a criação e animação correta da cena. No exemplo do pêndulo invertido, o mesmo foi construído com dois cubos (a base e a junta), um prisma (a haste) e uma esfera (o corpo equilibrado pelo pêndulo). Este modelo geométrico foi descrito a partir da base até o topo do mesmo nesta ordem, primeiro a base, depois a junta, depois a haste e por fim o corpo a ser equilibrado pelo pêndulo. Este tipo de construção de cena é baseado na estrutura DAG.

Cada parte do pêndulo possui obrigatoriamente um grupo com propriedades geométricas. É neste grupo que se define a forma, o material, a cor e a matriz de transformação local de cada uma das partes que compõe o pêndulo.

Algumas partes do pêndulo possuem um grupo com propriedades dinâmicas, que podem ser constantes ou variáveis. Estas constantes e variáveis também são descritas por nós não padrão do Open Inventor e servem para armazenar respectivamente constantes relativas à própria parte, tais como massa, comprimento, largura, altura, etc. e variáveis que representam as coordenadas generalizadas usadas na equação de dinâmica do pêndulo.

Foram associadas a cada parte que possui algum grau de liberdade variáveis que descrevem o tipo de grau de liberdade que a parte possui. Neste exemplo, a base possui um grau de liberdade que é um movimento de translação no eixo X e a haste possui um grau de liberdade que é um movimento de rotação no eixo Z. Estes graus de liberdade de cada parte estão representados respectivamente pelos seguintes trechos no arquivo:

```
#-- Coordenadas generalizadas associadas a base ----------------------
DEF coordgen0 CoordGen0 {
  fields [ SFString nome, SFString tipo, SFFloat valor,
  SFString nome_p, SFFloat valor_p]
  nome "x"                    #coordenada generalizada
  tipo "tx"                   #tipo de grau de liberdade (tranlação no eixo X)
  valor 0.0                   #valor inicial da coordenada generalizada
  nome_p "x_p"                #derivada da coordenada generalizada
  valor_p 0.0                 #valor inicial da derivada da coordenada generalizada
}

#-- Coordenadas generalizadas associadas a haste ----------------------
DEF coordgen0 CoordGen0 {
  fields [ SFString nome, SFString tipo, SFFloat valor,
  SFString nome_p, SFFloat valor_p]
  nome "theta"                #coordenada generalizada
  tipo "rz"                   #tipo de grau de liberdade (rotação no eixo Z)
  valor 0.5                   #valor inicial da coordenada generalizada
  nome_p "theta_p"            #derivada da coordenada generalizada
  valor_p 0.0                 #valor inicial da derivada da coordenada generalizada
}
```

Além da especificação do tipo do grau de liberdade, também é feita uma associação pelo nome à coordenada generalizada que é usada na equação de dinâmica do pêndulo, ou seja, a descrição da equação dinâmica do pêndulo é feita em função desta coordenada generalizada e de sua derivada, ambas descritas no mesmo grupo para cada parte.

Na descrição da equação de dinâmica também entram as constantes locais a cada parte do corpo (massa, comprimento, etc.) e constantes globais tais como a aceleração da gravidade.

Os três últimos grupos no final do arquivo servem para armazenar as constantes globais, as equações de dinâmica, e as leis de controle do sistema em simulação respectivamente. Cada um destes grupos possui tantos nós não padrões quantos forem necessários para definir as constantes globais, as equações de dinâmica e as leis de controle do robô em simulação. O exemplo a seguir mostra as definições de tais grupos para a simulação do pêndulo invertido.

```
#-- Constantes Globais ----------------------
DEF globais Group {
  DEF g0 G0 {
    fields [ SFString nome, SFFloat valor ]
    nome "g"   #Aceleração da gravidade
    valor 9.8  #valor da aceleração da gravidade
  }
}
#-- Equacoes de Dinâmica do Sistema ---------
DEF equacoes Group {
  DEF eq0 EQ0 {
    fields [ SFString nome_pp, SFString valor_pp ]
    nome_pp "x_pp"
    valor_pp "((Massa + massa)*g*theta - u)/(Massa*length)"
  }
  DEF eq1 EQ1 {
    fields [SFString nome_pp, SFString valor_pp ]
    nome_pp "theta_pp"
    valor_pp "(u - massa*g*theta)/Massa"
  }
}
#-- Equacoes de Controle do Sistema ---------
DEF controles Group {
  DEF crtl0 CRTL0 {
    fields [SFString nome, SFString valor ]
    nome "u"
    valor "298.15*theta + 60.68* theta_p + 163.08*x + 73.38*x_p"
  }
}
```

4.4 Conclusão

Neste capítulo foi apresentado como é feita a descrição geométrica do robô a partir do arquivo de descrição de cena da biblioteca Open Inventor. A descrição do comportamento dinâmico, a descrição da lei de controle do robô e a relação entre o modelo geométrico e as equações de dinâmica também foram incorporadas ao arquivo de descrição geométrica do robô através da inserção de nós não renderizáveis no arquivo de cena (arquivo iv). Desta forma as características do robô ficaram restritas a um único arquivo.

O capítulo 5 apresenta a integração do arquivo de descrição do robô (arquivo iv) com o Simulador, ou seja, apresenta como o Simulador utiliza o arquivo com extensão iv, como ele executa os cálculos de controle e de dinâmica, e como a cena é animada.

5 O simulador SimRobIV

Muitas simulações construídas para apoiar as fases de desenvolvimento e teste de robôs são restritas ao modelo do robô que está sendo testado e só exploram os aspectos cinemáticos do mesmo. Na simulação proposta neste trabalho, além da generalidade de modelos de robôs e da consideração dos aspectos dinâmicos do robô, implementou-se também um módulo de comandos que é responsável pelo envio de comandos ao robô e um módulo de controle que é responsável por simular um sistema de controle atuando na movimentação do robô. Como o módulo de comando e o módulo de controle fazem parte da simulação, esta possui um diferencial que é a possibilidade de ser utilizada para treinamento do usuário, que comanda o robô, e para teste da lei de controle. Sem a existência destes módulos a simulação serviria apenas para estudar a dinâmica do robô.

Neste capítulo é feito um detalhamento da implementação do protótipo do simulador proposto e batizado com o nome de SimRobIV. A intenção deste protótipo é mostrar que é possível construir uma simulação genérica, independente do modelo do robô, que leve em consideração a dinâmica do robô nos cálculos de sua movimentação, que permita que o mesmo seja comandado por um usuário e que apresente o resultado da simulação através de uma animação em 3D. Com a evolução deste protótipo pode-se construir uma ferramenta muito útil para treinar o usuário na tarefa de comandar o robô e para testar as leis de controle da movimentação do robô.

Este capítulo ficou então dividido em seções que apresentam a arquitetura do protótipo SimRobIV e alguns detalhes de sua implementação. A seção 5.1 apresenta os motivos para a escolha das ferramentas de desenvolvimento que foram utilizadas. A seção 5.2 apresenta a arquitetura proposta para o protótipo SimRobIV. A seção 5.3 apresenta detalhes de implementação do protótipo SimRobIV e a seção 5.4 mostra a execução do protótipo para duas simulações: a primeira delas é a simulação do pêndulo invertido e a segunda é a simulação de um braço articulado com dois graus de liberdade.

5.1 Paradigma de desenvolvimento e linguagem de programação

A estratégia adotada para a implementação do sistema de simulação foi utilizar o paradigma orientado a objeto para desenvolvimento. Como se adotou a biblioteca Open Inventor para a manipulação do modelo geométrico do robô e esta também foi projetada sob paradigma orientado a objeto e implementada na linguagem C++, decidiu-se implementar o simulador com a mesma linguagem, ou seja, C++.

A programação orientada a objetos proporciona facilidade de manutenção e principalmente de extensão do código. Além do fato da biblioteca Open Inventor estar implementada em C++, a implementação do simulador nesta linguagem tem a vantagem de torná-lo portável. Desta forma, fica claro a escolhas: paradigma orientado a objetos para desenvolvimento e linguagem C++ para implementação do simulador.

A lista a seguir resume todas as ferramentas utilizadas na construção do protótipo.

- Arquivos de entrada codificados em ASCII para passar para a simulação os comandos para o robô e os modelos geométrico, de controle e dinâmico do robô.
- Open Inventor - biblioteca gráfica escrita em C++ e orientada a objetos utilizada para a ler e manipular a descrição geométrica do robô e através dela gerar a animação em 3D.
- Protótipo da simulação escrito em C++ com chamadas às funções escritas em Python que efetuam os cálculos de controle e de dinâmica.
- Componente SoQt - componentes que auxiliam o desenvolvimento de aplicações gráficas traduzindo os eventos do sistema gerenciador de janelas para eventos internos da biblioteca Open Inventor, utilizado para apresentar a animação em 3D.

A figura 5-1 a seguir mostra como o simulador ficou dividido em módulos e em qual linguagem de programação cada módulo foi implementado.

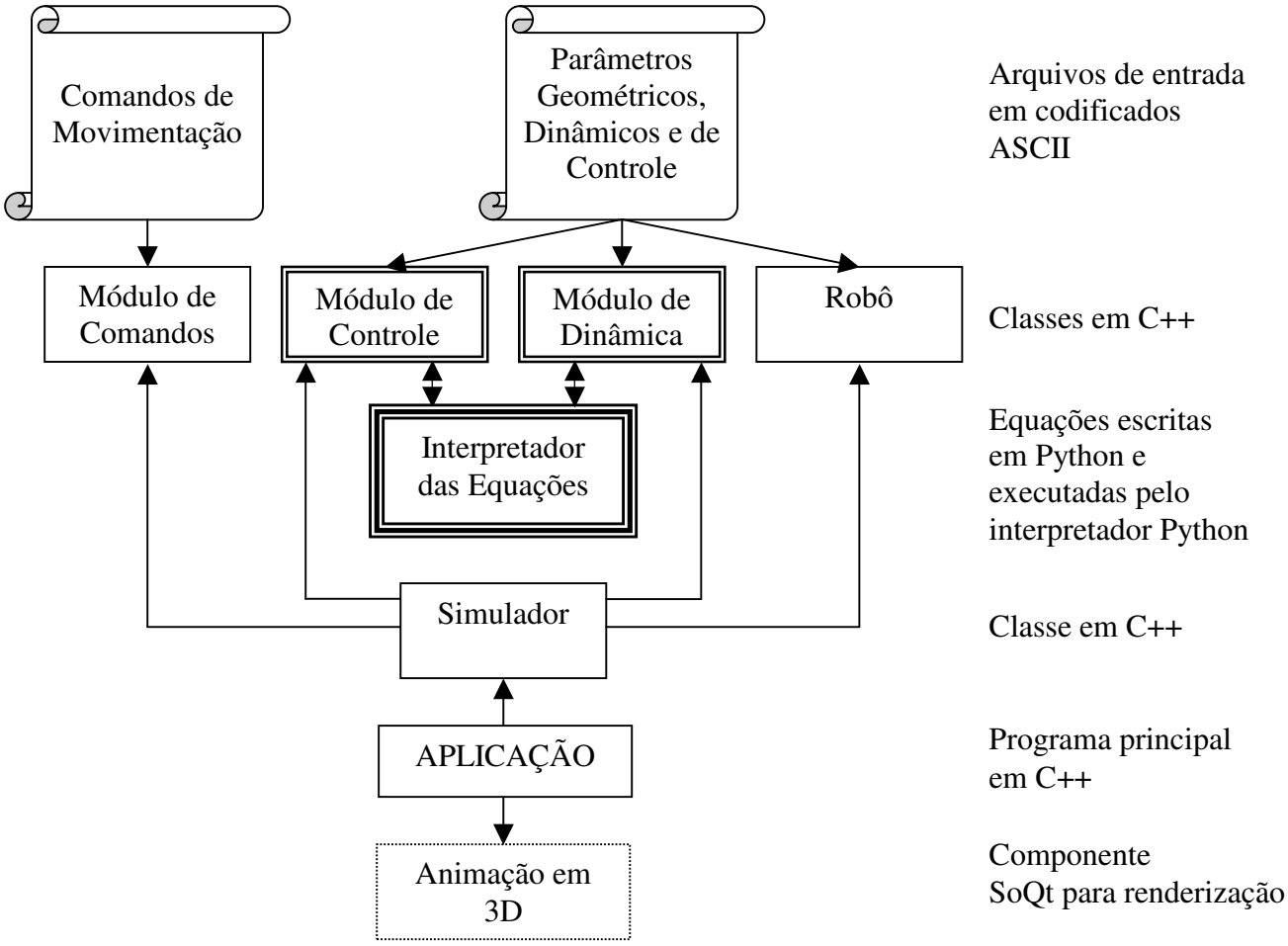


Figura 5-1 – Linguagens e ferramentas utilizadas para a construção do protótipo SimRobIV

A seguir são listadas todas a linguagens e bibliotecas que foram utilizadas para a implementação do protótipo.

Linguagens:

C++

Python 2.0

Bibliotecas:

Open Inventor 2.0

Qt 2.3

SoQt 1.0.2

5.2 Arquitetura

Para apresentar a arquitetura básica do protótipo pode-se recorrer ao ciclo da simulação apresentado no capítulo 2, dividindo-o em três grandes fases como mostra a figura 5-2.

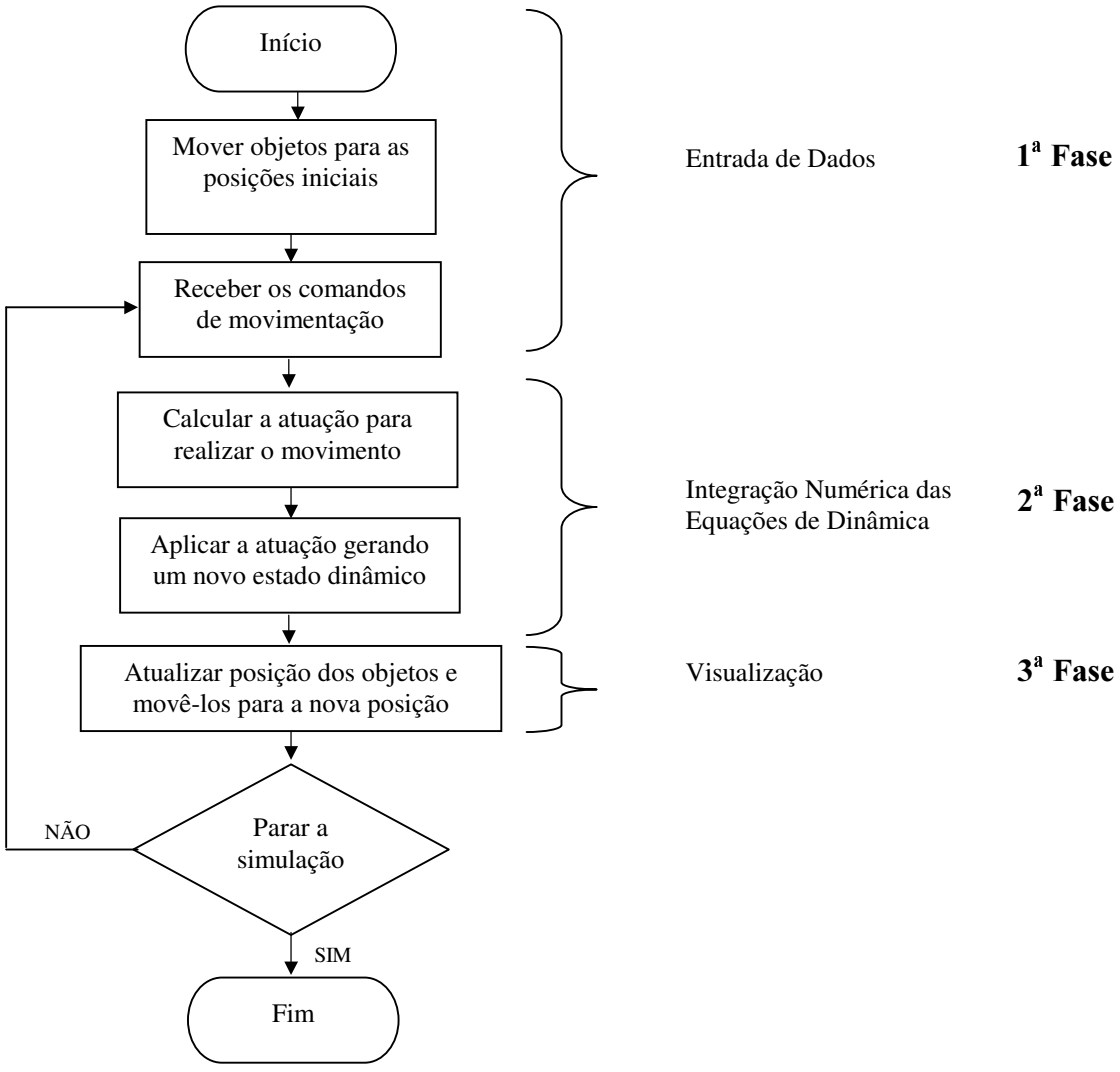


Figura 5-2 - Ciclo da simulação agrupado em Entrada de Dados, Integração e Visualização

A seguir a arquitetura do protótipo é detalhada segundo as fases: Entrada de Dados, Integração Numérica das Equações de Dinâmica e Visualização.

5.2.1 Entrada de Dados

A entrada de dados para a simulação é feita através de dois arquivos codificados em ASCII.

O primeiro é um arquivo chamado **Arquivo de Comandos** contendo em cada linha os comandos de posicionamento, para cada parte do robô que possui algum grau de liberdade, em um determinado instante de tempo. Em cada linha deste arquivo temos um valor para indicar o instante de tempo, a partir do qual o comando será executado, e um par de valores para indicar a posição e a velocidade para cada grau de liberdade que o robô a ser simulado pode apresentar, portanto para cada robô a quantidade de valores que compõem a linha do arquivo de comandos é da por *1+2N*, onde *N* é a quantidade de graus de liberdade do robô. Os valores de posição e velocidade para cada grau de liberdade podem representar valores lineares e valores angulares e devem ser todos escritos no mesmo sistema de medidas. A figura 5-3 mostra um exemplo de conteúdo do arquivo de comando para o sistema do pêndulo invertido que possui dois graus de liberdade.

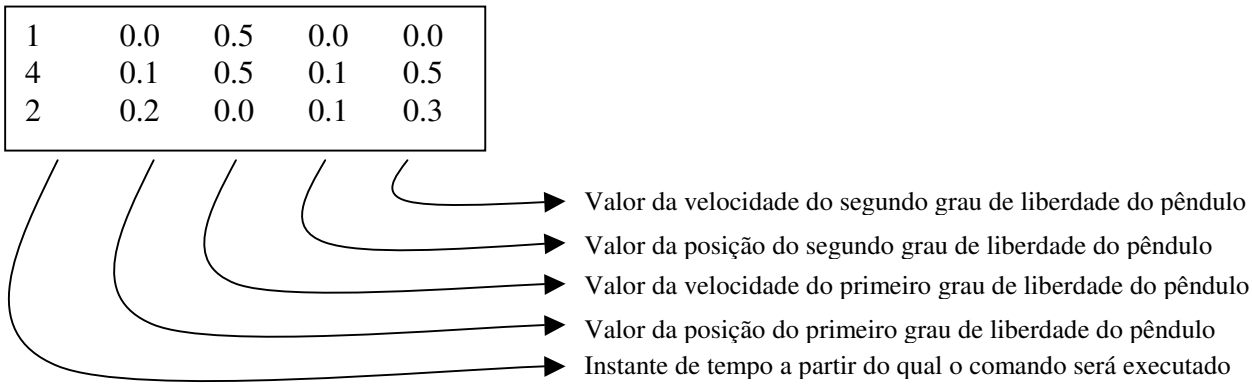


Figura 5-3 - Conteúdo do arquivo de comandos

A ordem com que os valores devem ser descritos, primeiro grau de liberdade, segundo grau de liberdade, e assim por diante, deve ser a mesma seguida para fazer a descrição dos parâmetros dinâmicos no arquivo de descrição do robô apresentado no capítulo 4. Cada grau de liberdade é representado por nó de coordenada generalizada no arquivo de descrição do robô. Para o usuário escrever corretamente o arquivo de comandos, deve seguir a mesma ordem em que as coordenadas generalizadas aparecem definidas no arquivo de descrição do robô, ou seja, o primeiro par de valores, posição e

velocidade, são relativos a primeira coordenada generalizada definida no arquivo de descrição do robô.

O segundo arquivo é chamado de **Arquivo de Descrição do Robô** e é escrito em um determinado padrão que é lido e reconhecido pela biblioteca Open Inventor, conforme a descrição feita no capítulo 4. Este arquivo pode estar codificado em ASCII ou em binário, porém, como este arquivo deve ser fornecido pelo usuário que deseja realizar a simulação, o usuário o fornecerá em código ASCII. Neste arquivo o usurário faz a descrição geométrica, informa qual é o comportamento dinâmico (equações de dinâmica) e qual é a lei de controle que atua na movimentação do robô.

Durante a fase de inicialização da simulação, além da leitura destes dois arquivos de entrada, **Arquivo de Comandos** e **Arquivo de Descrição do Robô**, outros arquivos intermediários são gerados para dar apoio a algumas funções durante a simulação.

A figura 5.4 apresenta o diagrama de inicialização da simulação com os arquivos de entrada e os arquivos intermediários gerados nesta fase. O Arquivo de Comando possui a extensão cmd e o Arquivo de Descrição do Robô possui a extensão iv. No exemplo da figura 5-4 os arquivos de entrada estão representados por **robo.cmd** e **robo.iv**.

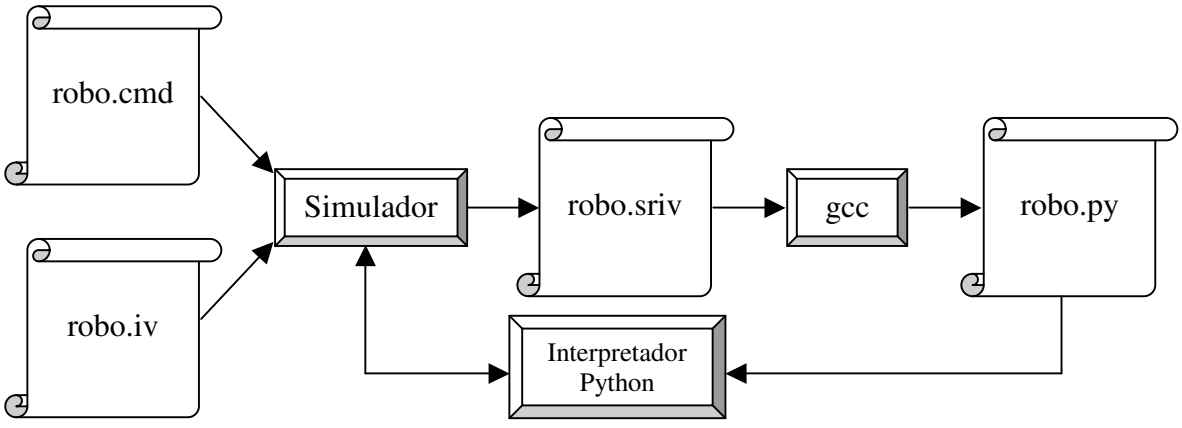


Figura 5-4 - Arquivos de entrada e arquivos intermediários auxiliares à simulação

Primeiramente, o Simulador lê o arquivo com extensão **iv,** extrai deste as equações de controle e de dinâmica e cria um relacionamento entre: as partes do robô; as coordenadas generalizadas, com as quais as equações de dinâmica foram escritas; e o vetor de estados do robô.

Internamente a simulação mantém o **vetor de estados do robô** e uma **tabela de associação** entre a **parte do robô,** o **tipo de grau de liberdade** e o **vetor de estados**. Os esquemas das figuras 5-5 e 5-6 a seguir mostram como seriam o **vetor de estado** e a **tabela de associação**, para um robô com *M* partes e *N* graus de liberdade.

Cada grau de liberdade corresponde a uma coordenada generalizada do robô. Para cada grau de liberdade o vetor de estado armazena o valor da coordenada generalizada e o valor da derivada desta coordenada generalizada. Para um robô com *N* graus de liberdade o vetor de estados teria um tamanho de *2N* posições para armazenar em cada posição um dado do tipo *double* em C++.

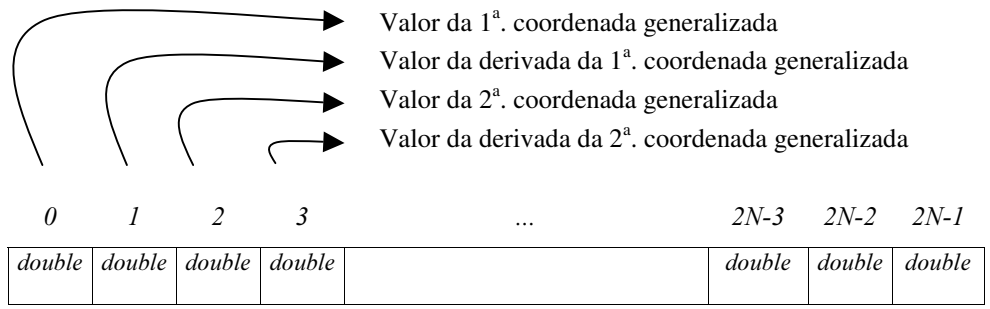


Figura 5-5 - Vetor de estados de um Robô com N graus de liberdade

Nem todas as partes do robô possuem coordenadas generalizadas associadas, somente as partes que possuem um movimento (um grau de liberdade) é que possuem a coordenada generalizada associada. Como o robô é construído por sub partes hierarquicamente relacionadas, basta descrever a movimentação de uma sub-parte para que todas as sub-partes posteriores herdem este movimento. Por exemplo, em um braço mecânico cuja base possui um movimento de translação no eixo X, todas as outras partes posteriores diretamente acopladas à base, também sofrem a mesma translação aplicada à base, neste caso a base é uma parte que possui uma coordenada generalizada associada.

A tabela de associação é uma estrutura interna ao Simulador e mantém a informação de qual elemento do vetor de estados está associado à qual parte do robô.

Número da Parte do Robô: 0 a (M-1).	*Tipo de Grau de Liberdade: TX, TY ou TZ – Translação em um dos eixos. RX, RY ou RZ – Rotação em um dos eixos.*	*Índice do Vetor de Estados que armazena o valor da Coordenada Generalizada correspondente: 0 a (2N-1).*
1	TX	0

Figura 5-6 - Tabela de associação Parte do Robô/Tipo de Grau de Liberdade/Vetor de Estados

Após a leitura dos arquivos de entrada **robo.cmd** e **robo.iv**, dois arquivos auxiliares são criados pelo Simulador para serem utilizados durante os cálculos da simulação. Estes arquivos são: **robo.sriv** e **robo.py**.

O arquivo **robo.sriv** mantém o relacionamento entre as coordenadas generalizadas e o vetor de estados do robô, as leis de controle e as equações de dinâmica. Este arquivo é passado para um pré-compilador para que ele faça algumas substituições de símbolos (*toquens)* e reescreva um arquivo com extensão py. O arquivo reescrito em conformidade com a sintaxe da linguagem Python possui as equações escritas em função dos elementos do vetor de estado do robô. A figura 5-7 mostra um exemplo de como pode ser feita a substituição de *toquens* usando o pré-compilado gcc.

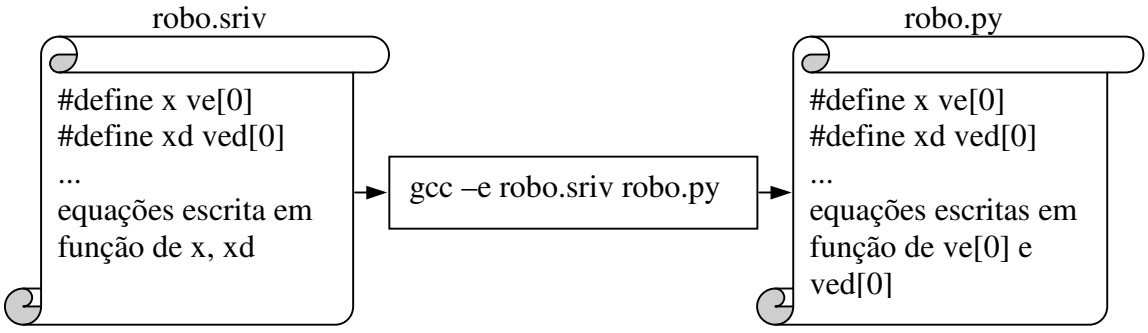


Figura 5-7 Utilização do gcc com o parâmetro -e que executa a pré-compilação

O arquivo robo.py gerado contém as funções que o interpretador Python utiliza para fazer os cálculos de controle e de dinâmica.

O exemplo da figura 5-8 mostra o conteúdo do arquivo robo.sriv para a simulação do pêndulo. Pode-se notar que a primeira parte do arquivo contém a definição de coordenadas generalizadas em função de elementos do vetor de estados.

```
#define interpretador #! /usr/local/bin/pyhton
#define import import math
#define x ve[0]
#define xd ved[0]
#define x_p ve[1]
#define x_pd ved[1]
#define theta ve[2]
#define thetad ved[2]
#define theta_p ve[3]
#define theta_pd ved[3]
#define x_pp vep[1]
#define theta_pp vep[3]
#define t1 atua[0]
#define t2 atua[1]
interpretador
import
g=10
Massa=2
length=0.5
massa=0.1

def controle (ve,ved):
       atua=list()
       py_elemento=0
       while py_elemento < 2:
              atua.append(0)
              py_elemento=py_elemento+1
       t1=298.15*(theta-thetad) + 60.69*(theta_p-theta_pd) + 163.09*(x-xd) +
       73.39*(x_p-x_pd)
       t2=298.15*(theta-thetad) + 60.69*(theta_p-theta_pd) + 163.09*(x-xd) +
       73.39*(x_p-x_pd)
       return (atua)

def dinamica (ve,atua):
       vep=list()
       py_elemento=0
       while py_elemento < len(ve):
              vep.append(0)
              py_elemento=py_elemento+1
       x_pp=(t1 - massa*g*theta)/Massa
       theta_pp=((Massa + massa)*g*theta - t2)/(Massa*length)
       i=0
       while i<len(ve):
              vep[i]=ve[i+1]
              i=i+2
       return (vep)
```

Figura 5-8 Conteúdo do arquivo sriv para a simulação do pêndulo invertido

A figura 5-9 mostra o conteúdo do arquivo robo.py para a simulação do pêndulo invertido. Neste arquivo existe duas funções a primeira chamada **controle** e segunda chamada **dinamica**.

```
#! /usr/local/bin/pyhton
import math
g=10
Massa=2
length=0.5
massa=0.1

def controle (ve,ved):
        atua=list()
        py_elemento=0
        while py_elemento < 2:
                atua.append(0)
                py_elemento=py_elemento+1
        atua[0]=298.15*(ve[2]-ved[2]) + 60.69*(ve[3]-ved[3]) + 163.09*(ve[0]-ved[0])
+ 73.39*(ve[1]-ved[1])
        atua[1]=298.15*(ve[2]-ved[2]) + 60.69*(ve[3]-ved[3]) + 163.09*(ve[0]-ved[0])
+ 73.39*(ve[1]-ved[1])
        return (atua)

def dinamica (ve,atua):
        vep=list()
        py_elemento=0
        while py_elemento < len(ve):
                vep.append(0)
                py_elemento=py_elemento+1
        vep[1]=(atua[0] - massa*g*ve[2])/Massa
        vep[3]=((Massa + massa)*g*ve[2] - atua[1])/(Massa*length)
        i=0
        while i<len(ve):
                vep[i]=ve[i+1]
                i=i+2
        return (vep)

def dinamica (ve,atua):
       vep=list()
       py_elemento=0
       while py_elemento < len(ve):
              vep.append(0)
              py_elemento=py_elemento+1
       x_pp=(t1 - massa*g*theta)/Massa
       theta_pp=((Massa + massa)*g*theta - t2)/(Massa*length)
       i=0
       while i<len(ve):
              vep[i]=ve[i+1]
              i=i+2
       return (vep)
```

Figura 5-9 Conteúdo do arquivo py para a simulação do pêndulo invertido

Posteriormente o interpretador Python é evocado com o arquivo **robo.py** passado como parâmetro. Desta forma as funções **controle** e **dinamica** ficam disponíveis em memória durante toda a simulação.

Ao término desta primeira fase ou fase de inicialização, a simulação já possui o modelo geométrico do robô, o vetor de estado, a tabela de associação entre uma parte do robô e o vetor de estado e as funções, de controle e dinâmica, escritas em Python. Quando o usuário solicita que a simulação seja iniciada é dado início a um ciclo com duas tarefas:

- cálculo do novo vetor de estado;
- atualização das propriedades geométricas (posição e orientação) das partes do robô que estão associadas ao vetor de estado.

Estas duas tarefas, que estão detalhadas nas sessões 5.2.2 e 5.2.3 respectivamente, são controladas pela simulação através de uma classe que simula uma fila circular de eventos. A biblioteca Open Inventor possui uma classe chamada SoTimerSensor com a qual é possível encapsular eventos ou funções do usuário, para que estes sejam disparados ciclicamente em um determinado intervalo de tempo. O exemplo da figura 5-10 mostra como a função de cálculo da simulação e a função de animação são encapsuladas pela classe SoTimerSensor.

```
//Encapsulamento do evento de cálculo
GLOBAL_sensorSimula = new SoTimerSensor(executaSimulacao,GLOBAL_sim);
GLOBAL_sensorSimula->setBaseTime(0.0);
GLOBAL_sensorSimula->setInterval(SbTime(0.5));
GLOBAL_sensorSimula->schedule();

//Encapsulamento do evento de animação
GLOBAL_sensorAnima = new SoTimerSensor(anima,GLOBAL_sim->recuperaRobo());
GLOBAL_sensorAnima->setBaseTime(0.0);
GLOBAL_sensorAnima->setInterval(SbTime(1.0));
GLOBAL_sensorAnima->schedule();
```

Figura 5-10 Inserção do eventos de cálculo e animação em uma fila circular de eventos

No exemplo da figura 5-10, o método `setInterval` da classe SoTimerSensor define a periodicidade que o evento ou função do usuário é disparado, já o método `schedule` insere o objeto, que encapsula o evento ou função do usuário, em fila de

eventos que é mantida e processada no ciclo infinito da aplicação que utiliza a biblioteca Open Inventor.

5.2.2 Integração das equações de dinâmica

Durante a fase de inicialização da simulação, a primeira operação a ser realizada é a de leitura do arquivo de descrição do robô. Esta leitura, além de criar em memória objetos que representam graficamente as partes do robô, também extrai as equações de controle e de dinâmica que estão definidas neste arquivo. Com estas equações as funções de controle e de dinâmica são escritas em Python e utilizadas durante todo o ciclo da simulação.

Uma vez terminada a primeira fase ou fase de inicialização da simulação, o usuário pode dar início a simulação. A simulação pode ser resumida na seguinte seqüência de passos:

- Cálculo da atuação (feito pelo módulo de controle);
- Cálculo do novo estado dinâmico (feito pelo módulo de dinâmica);

A execução deste dois passos é controlada pela fila de eventos da biblioteca Open Invertor.

Para a implementação dos módulos de controle e de dinâmica, apresentados no capítulo 3, fez-se a opção pelas linguagens C++ e Python, pois estes módulos devem ser capazes de interpretar as equações de dinâmica e de controle fornecidas pelos usuários para cada simulação. Estas equações, tanto as de dinâmica quanto as de controle, são próprias do modelo de robô que se deseja simular, portanto para cada simulação estas equações precisam ser interpretadas. Existiriam então duas saídas, a primeira delas é escrever ou utilizar um interpretador de equações em C++ e a segunda é usar uma linguagem interpretada para escrever estas equações. Como o foco é a simulação e não a escrita de um interpretador de equações de dinâmica e de controle, decidiu-se utilizar uma linguagem interpretada que tivesse um bom desempenho. Novamente fez-se uma análise em um conjunto reduzido de linguagens interpretadas (Java, Lua, Perl e Python) e escolheu-se Python pela facilidade de programação e pela existência de uma API para trocar dados com um programa escrito em C++. A figura 5-11 mostra como os módulos de controle e de dinâmica foram implementados.

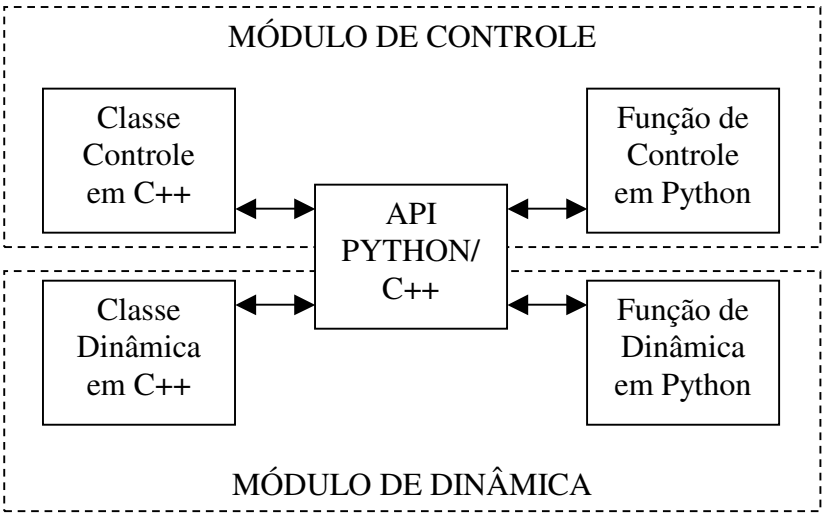


Figura 5-11 Esquema de implementação dos módulos de controle e de dinâmica em C++/Python

O cálculo da atuação, feito pelo módulo de controle, se dá da seguinte forma:

- O módulo de controle (uma classe escrita em C++) passa para a função de controle (função escrita em Python), via a API Python/C++, um vetor contendo os valores das posições atual e desejada para cada parte do robô;
- A função de controle (função escrita em Python) devolve, via a API Python/C++, um vetor contendo os valores da força/torque que devem ser aplicados a cada parte do robô para que ele realize o movimento. Este vetor devolvido, também chamado de vetor atuação, é calculado de acordo com a lei de controle que a função em Python mantém.

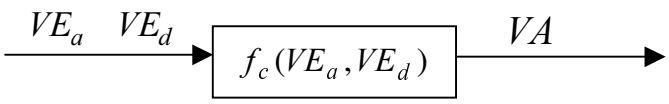


Figura 5-12 Representação da função de controle escrita em Python

Onde:

f_c representa a função de controle escrita em Python;

VE_a representa o vetor de estado atual;

VE_d representa o vetor de estado desejado;

VA representa o vetor atuação;

Um exemplo do conteúdo da função de controle pode ser visto na figura 5-9.

O cálculo do novo estado dinâmico, feito pelo módulo de dinâmica, se dá da seguinte forma:

- O módulo de dinâmica (uma classe escrita em C++) passa para a função de dinâmica (função escrita em Python), via a API Python/C++, um vetor contendo os valores da força/torque que devem ser aplicados a cada parte do robô (vetor atuação) e um vetor contendo os valores das posições atuais de cada parte do robô;
- A função de dinâmica (função escrita em Python) devolve, via a API Python/C++, um vetor contendo valores que representam a derivada dos valores armazenados vetor de estado

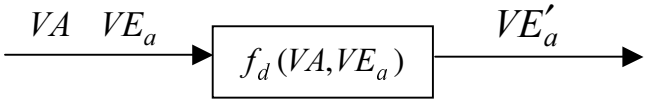


Figura 5-13 Representação da função de dinâmica escrita em Python

Onde:

f_d representa a função de dinâmica escrita em Python;

VA representa o vetor atuação;

VE_a representa o vetor de estado atual;

VE'_a representa a derivada vetor de estado atual;

De posse do vetores VE_a e VE'_a passa-se então para a integração numérica para efetuar o cálculo do novo vetor de estados VE_n. O novo vetor de estado é utilizado como vetor de estado atual para o próximo ciclo dos cálculos e é utilizado também para gerar a animação.

A integração das equações de dinâmica é realizada pela classe SRIV_Integrador que cria o integrador padrão (método de Euler) da simulação. O valor do passo de integração é informado pelo usuário antes do início da simulação através da janela de parâmetros, como pode ser visto na figura 5-14.

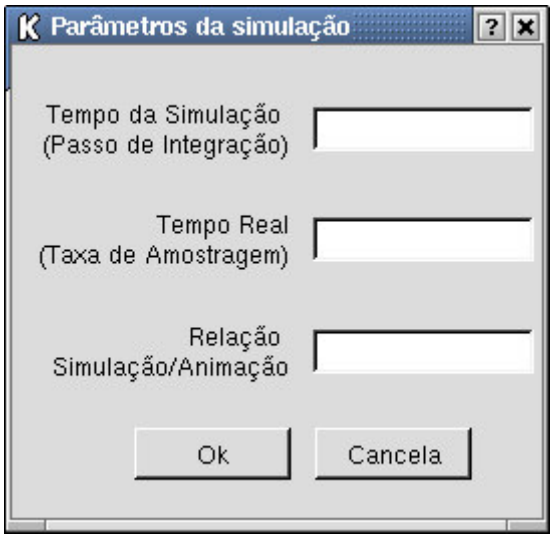


Figura 5-14 Janela de definição de parâmetros da simulação

O próximo passo é a utilização do novo vetor de estado para gerar a animação.

5.2.3 Visualização

Inicialmente, a visualização da animação foi realizada em um componente Xt, ou seja, um componente customizado para traduzir os eventos do sistema X Windows para eventos que a biblioteca Open Inventor possa tratar.

Criou-se uma aplicação console com um objeto janela da classe SoXt, que era a janela principal da aplicação. Logo em seguida, criou-se um objeto janela de visualização da cena com a classe `SoXtRenderArea`, sendo que este era o objeto responsável pela renderização do grafo da cena ao qual era associado. Após a criação da janela principal e da janela de renderização, solicitava-se a apresentação das mesmas, através do método `show()`. Por último a janela principal evocava o ciclo infinito de tratamento de eventos, onde os eventos do sistema X Windows são traduzidos para eventos internos da biblioteca Open Inventor. A figura 5-15 mostra a seqüência destas operações.

```
Widget janela = SoXt::init();
SoXtRenderArea * areaRender = new SoXtRenderArea(janela);
areaRender->show();
SoXt::show(janela); //Apresenta janela principal
SoXt::mainLoop();
```

Figura 5-15 Sequência de comandos de inicialização de uma aplicação usando o componente SoXt

Posteriormente, para poder introduzir mais facilmente outros elementos na interface gráfica com o usuário, adotou-se um componente clone do SoXt chamado SoQt, que é um conjunto de componentes que auxiliam o desenvolvimento de aplicações com uma interface gráfica traduzindo os eventos do sistema gerenciador de janelas para eventos internos da biblioteca Open Inventor. Os componentes SoQt possuem a mesma funcionalidade dos componentes SoXt, porém são escritos com a biblioteca Qt. A vantagem de se utilizar a biblioteca Qt [QT] para desenvolvimento de aplicações gráficas, é a possibilidade de trocar de sistema operacional sem a necessidade de reescrever toda a aplicação bastando somente recompilá-la para a nova plataforma. A biblioteca Qt permite que aplicações com interface gráfica sejam escritas de sem a dependência do sistema gerenciador de janelas.

Conforme visto na sessão 5.2.2, durante a simulação são realizados os cálculos de controle e de dinâmica que fazem com que o vetor de estado do robô seja atualizado a cada intervalo de tempo. Este intervalo de tempo é definido pelo passo de integração. A animação da cena se dá pela transferência dos valores armazenados no novo vetor de estado do robô para as respectivas partes do robô que estão associadas a este vetor de estado. A transferência dos valores do vetor de estado para os objetos da cena pode ser toda a vez que o vetor de estado é atualizado ou pode ser realizada em intervalos de tempo maiores, isto vai depender da relação Simulação/Animação que o usuário informa na janela de parâmetros da simulação (ver figura 5-14). Esta relação define de quantas execuções de cálculos de simulação devem ser feitas para cada execução de uma animação.

5.3 Detalhes de implementação

O modelo de classes apresentado na figura 5-16 a seguir mostra as classes divididas em dois pacotes, o primeiro deles composto somente por classes do problema e o segundo composto por classes de interface com o usuário.

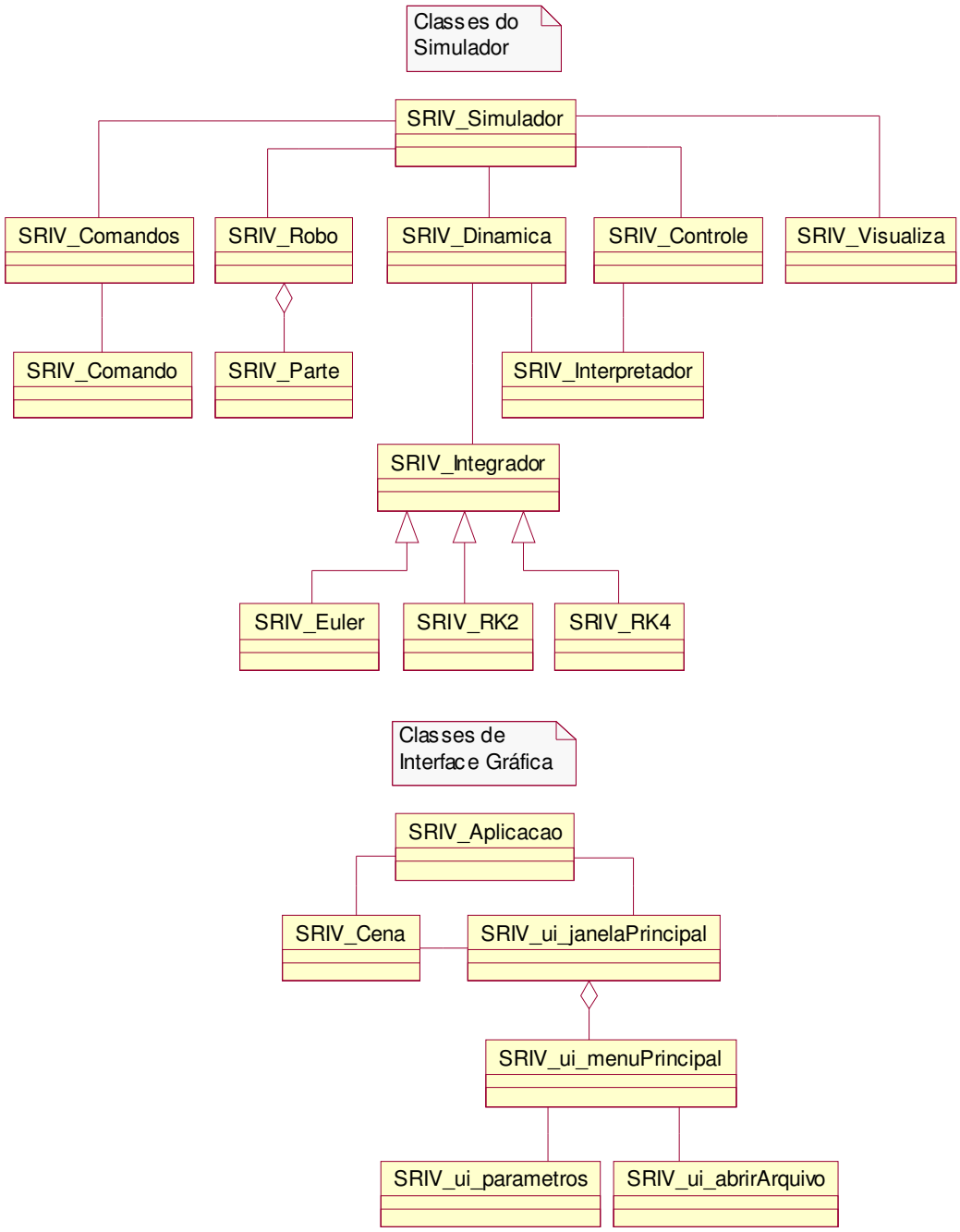


Figura 5-16 Modelo de classes do ambiente de simulação de dinâmica e controle de robôs

A lista a seguir apresenta uma descrição resumida sobre a funcionalidade das classes do problema que estão no modelo de classe da figura 5-16.

A classe **SRIV_Simulador** é responsável pelo gerenciamento da simulação, ou seja, pela comunicação entre as classes Comando, Controle, Dinâmica e Robô.

A classe **SRIV_Robo** é responsável por carregar o modelo geométrico, o modelo dinâmico e a lei de controle do robô a ser simulado e por transferir os valores do vetor de estado do robô para o grafo da cena.

A classe **SRIV_Comandos** enfileira os comandos de movimentação passados ao robô através de um arquivo texto.

A classe **SRIV_Controle** é responsável por calcular a atuação (torque ou força generalizada). Este cálculo é feito com o auxílio da classe **SRIV_Interpretador**.

A classe **SRIV_Dinamica** é responsável por calcular o novo estado do robô, ou seja, calcular quais são os novos valores de posição e velocidade de cada parte que compõe o robô. Este cálculo é realizado com o auxílio da classe **SRIV_Interpretador** e da classe **SRIV_Integrador**.

A classe **SRIV_Interpretador** é responsável por prover o cálculo da lei de controle e da dinâmica para as classe **SRIV_Controle** e **SRIV_Dinamica**, através da troca de dados com o interpretador Python.

A classe **SRIV_Integrador** é responsável por prover um método de integração para a classe **SRIV_Dinamica**.

No apêndice D é apresentado o diagrama de seqüência (padrão UML) do simulador. Neste diagrama de seqüência é possível acompanhar e conhecer toda a troca de mensagens entre classes. O diagrama apresenta as mensagens na ordem em que elas ocorrem no sistema sobre uma linha de tempo que se estende do topo, início desta linha, até a base, fim desta linha.

5.4 A aplicação

A aplicação executa a simulação e apresenta os resultados da mesma através da animação de uma cena estática que foi criada no arquivo de descrição do robô.

A janela principal da aplicação é apresentada vazia e com um *menu* como mostra a figura 5-17 a seguir.

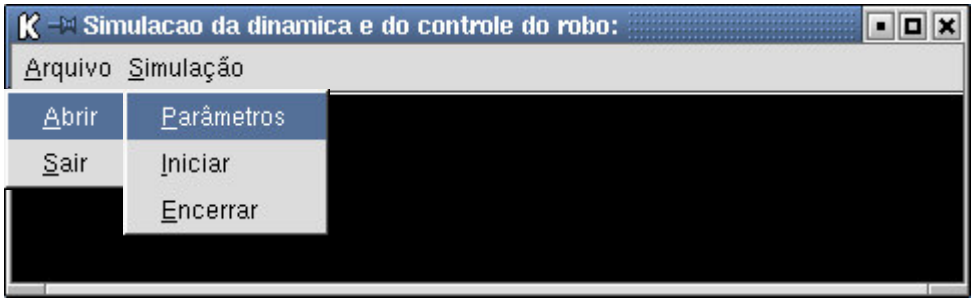


Figura 5-17 - Janela principal da aplicação

O usuário pode escolher o arquivo contendo a descrição do robô através da janela de seleção de arquivo com extensão iv, como mostra a figura 5-18 a seguir.

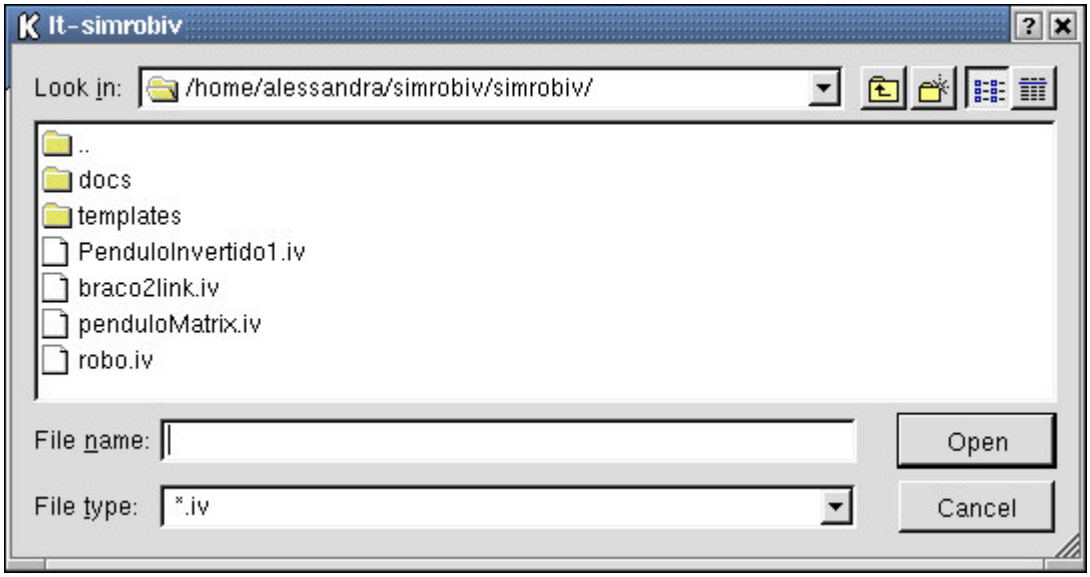


Figura 5-18 - Janela de seleção do arquivo de descrição do robô

Antes de dar início à simulação, o usuário pode definir o **passo de integração**, a **taxa de amostragem**, que é o valor de quanto em quanto tempo real é feito o cálculo da simulação e a **relação simulação/animação**, que define quantos ciclos de simulação são executados para cada ciclo de animação. A figura 5-19 a seguir mostra a janela de definição destes parâmetros.

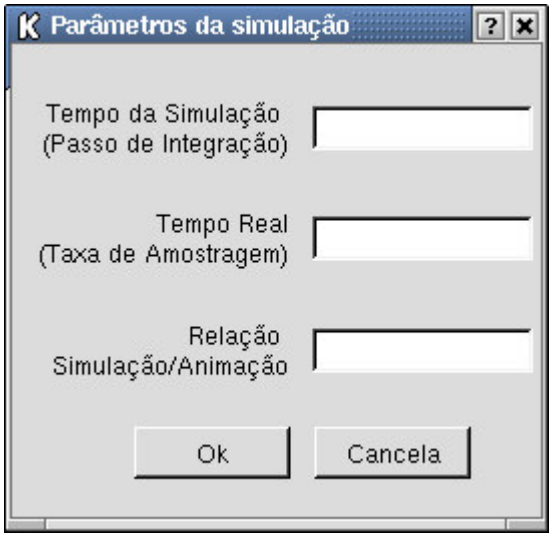


Figura 5-19 - Janela de definição de parâmetros da simulação

Após selecionar o arquivo de descrição do robô a cena é renderizada e apresentada com o robô em sua condição inicial. Deste momento em diante o usuário poder dar início a simulação através do item Inciar do *menu* Simulação. As figuras 5-20 e 5-21 mostram o pêndulo invertido e o braço de dois *links* e sua condição inicial.

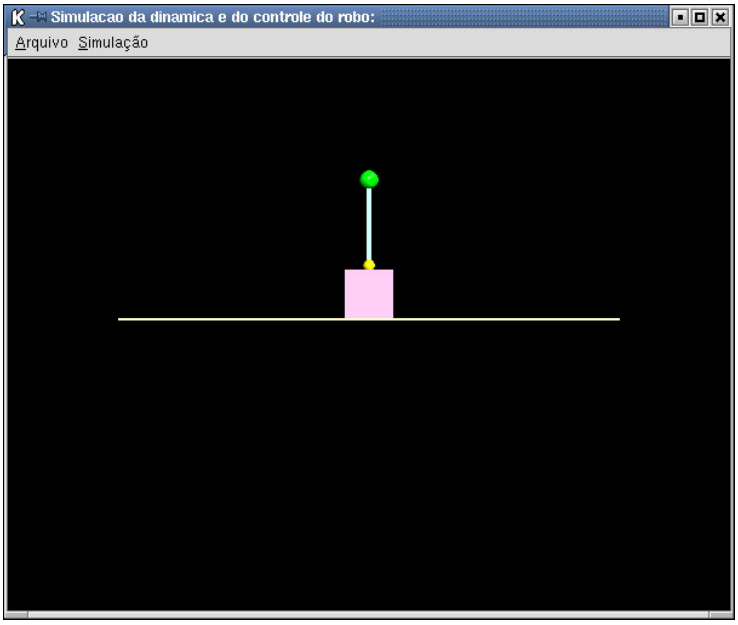


Figura 5-20 - Cena renderizada com o pêndulo invertido em seu estado inicial

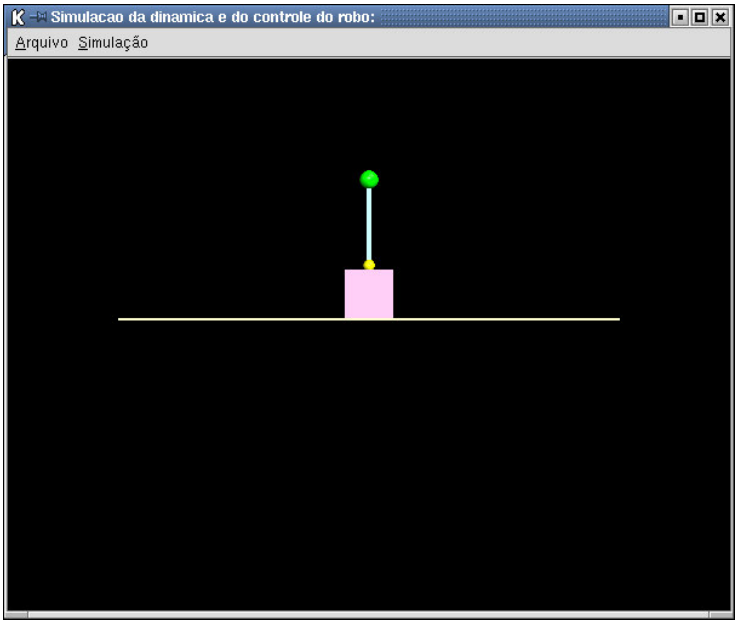


Figura 5-21 - Cena renderizada com o braço de dois links em seu estado inicial

5.5 Conclusão

O protótipo desenvolvido neste trabalho serve para reforçar a idéia de que é possível criar um sistema de simulação que considere a dinâmica e o controle de vários tipos de robôs de forma parametrizada. Este tipo de sistema pode ajudar muito na fase de projeto de robôs e na fase de teste das leis de controle. Também é possível treinar o usuário na operação do robô, uma vez que o sistema leva em consideração o recebimento de comando de movimentação para o robô.

6 Conclusões e Perspectivas futuras

6.1 Conclusões

A grande vantagem desta proposta de simulação é a possibilidade de simular a ação de comando do robô, pois adia a necessidade de se construir um protótipo físico para testes das leis de controle. Na simulação proposta é possível testar se a lei de controle está adequada, através do envio de comandos de movimentação ao robô e através da análise do resultado apresentado pela animação em 3D gerada.

O protótipo da simulação serviu para mostrar que é possível realizar simulações para modelos distintos de robôs.

Como contribuições deste trabalho podemos listar:

- A consideração do sistema de controle durante a execução da simulação
- A possibilidade de simularmos robôs distintos;
- A utilização do formato de arquivo padronizado da biblioteca Open Inventor para a descrição geométrica, de controle e de dinâmica do robô. Como o formato de arquivo da biblioteca Open Inventor serviu com modelo para o formato de arquivo de VRML, fica fácil migrar a simulação para um ambiente web ou de realidade virtual.

6.2 Perspectivas futuras

Como é um protótipo ainda necessita de algumas extensões para torná-lo uma ferramenta de trabalho mais complenta em testes de robôs. A segui são apresentadas as extensões que podem ser feitas de imediato para tornar este protótipo mais versátil:

- Implementar um editor com uma interface gráfica 3D, para que o arquivo de descrição das características geométricas, dinâmicas e de controle do robô seja gerado e mantido mais facilmente pelo usuário final de forma mais amigável;
- Implementação ou utilização de um módulo de detecção e tratamento do evento de colisão;
- Geração automática das equações de dinâmica
- Utilização de um dispositivo de entrada mais elaborado, como por exemplo, um *joystick*, que facilite o treinamento do usuário no comando do robô.

Apêndices

A - Ícones utilizados na representação dos nós do grafo de uma cena em Open Inventor.

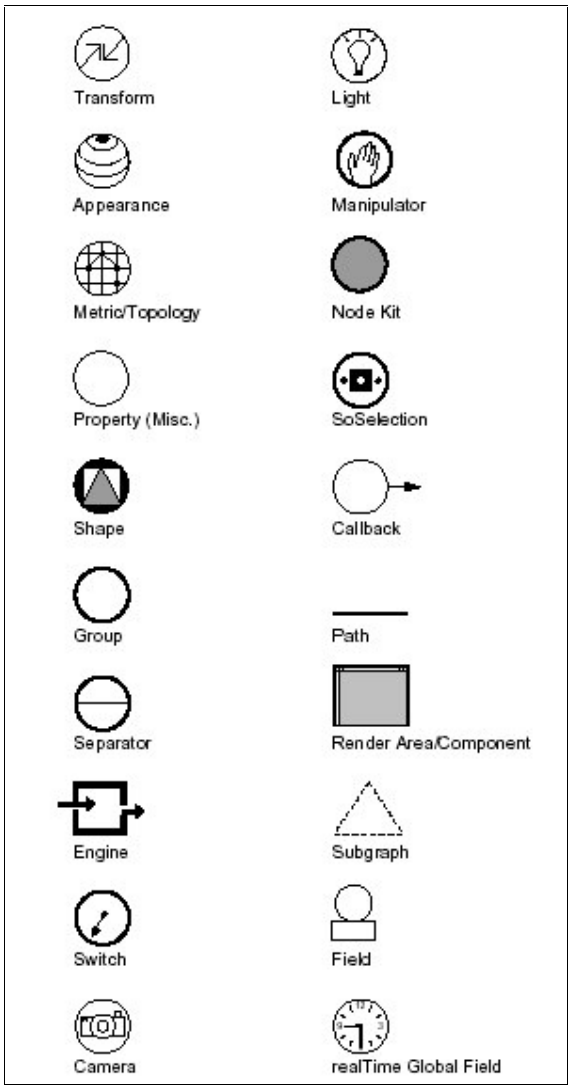


Figura A-1 - Símbolos do grafo de cena

B - O modelo de classe da biblioteca Open Inventor.

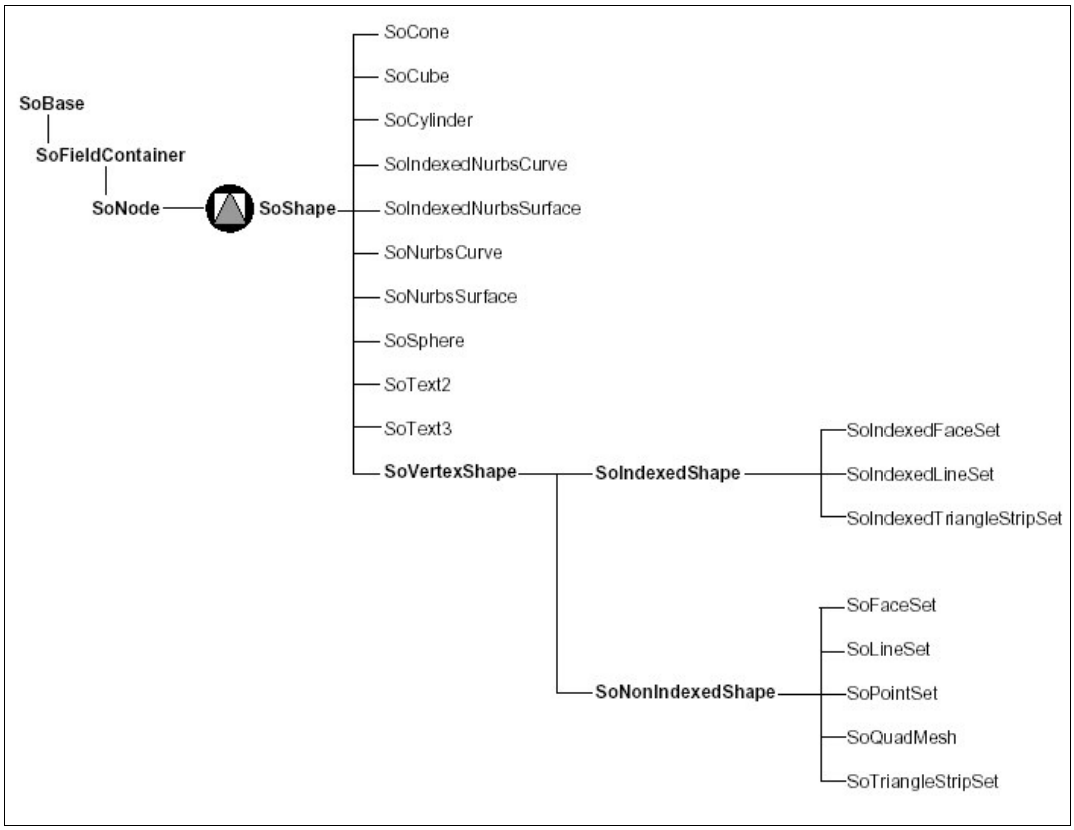


Figura B-1 - Classes que representam os nós de forma

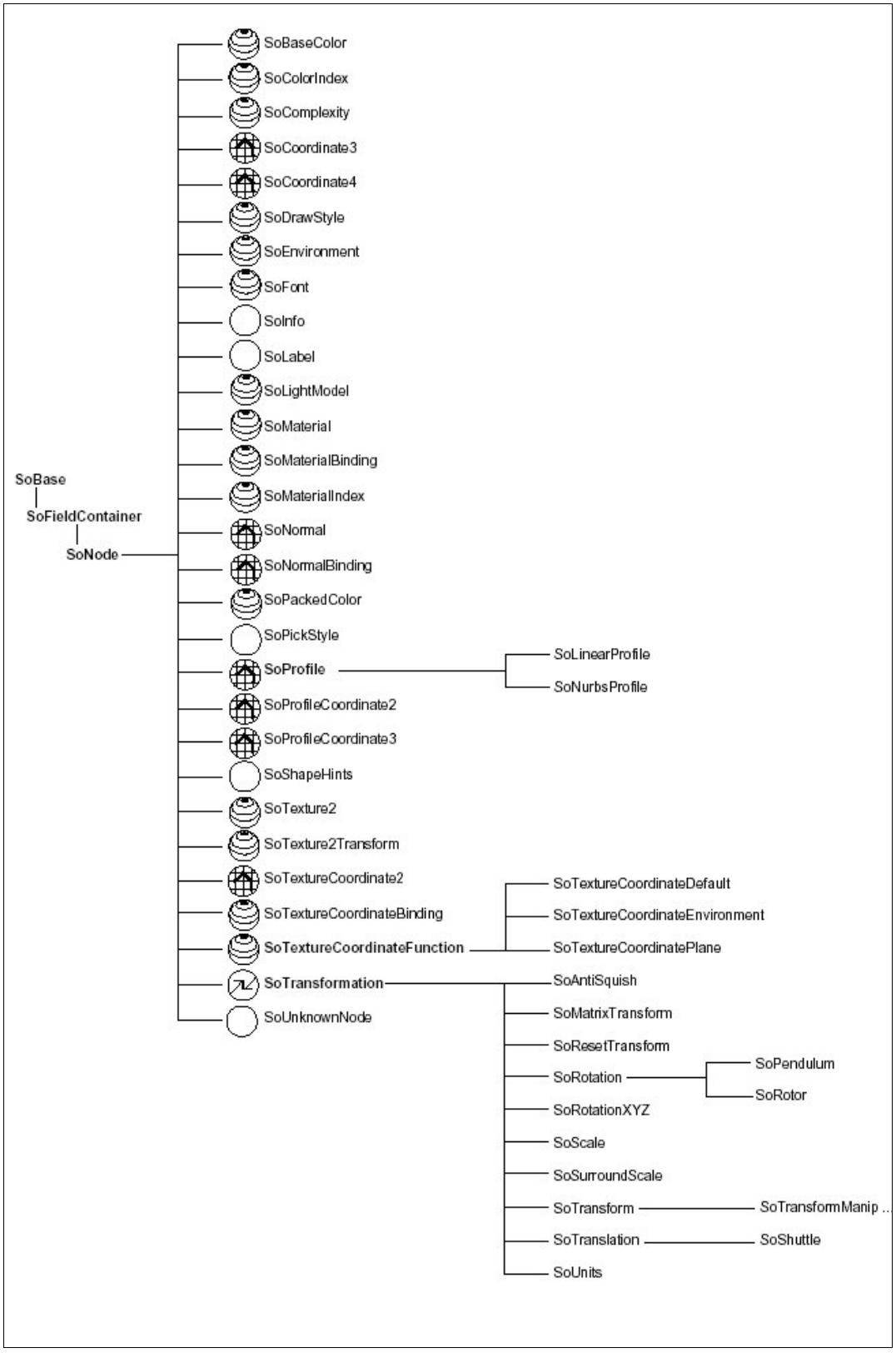


Figura B-2 - Classes que representam os nós de propriedade

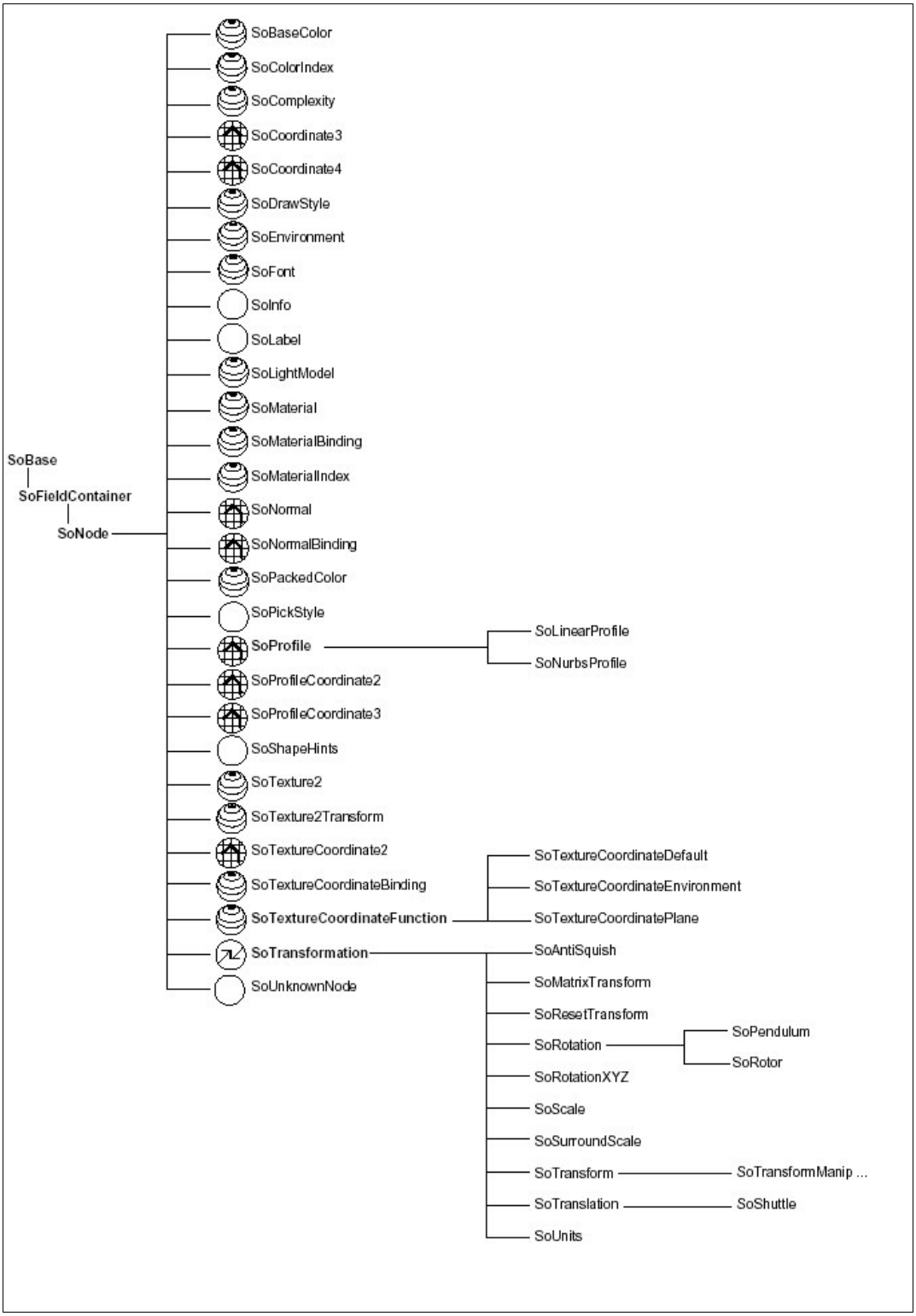


Figura B-3 - Classes que representam os nós de propriedade

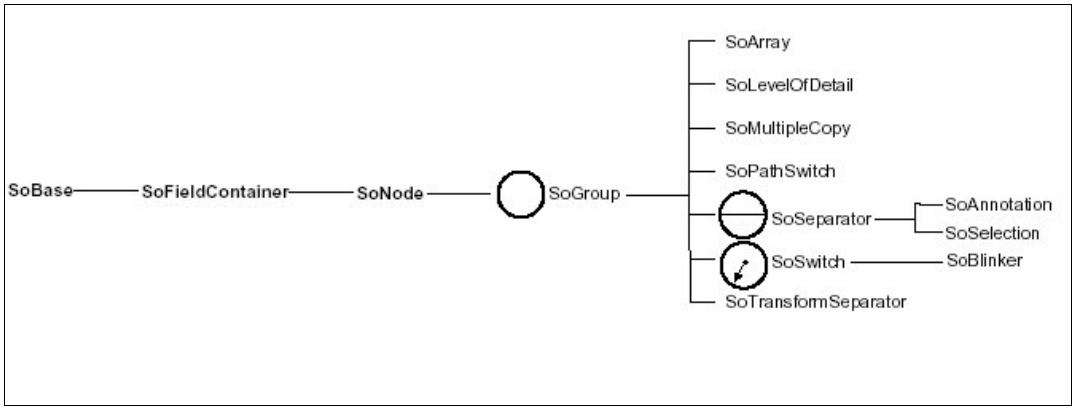


Figura B-4 - Classes que representam os nós de grupos

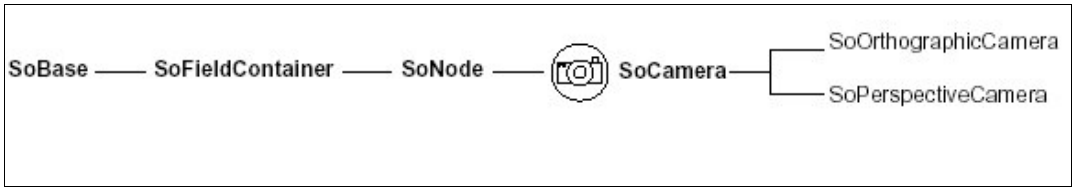


Figura B-5 - Classes que representam os nós de câmera

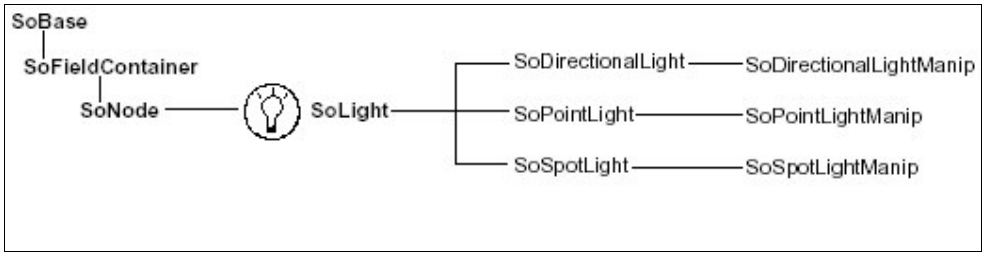


Figura B-6 - Classes que representam os nós de luz

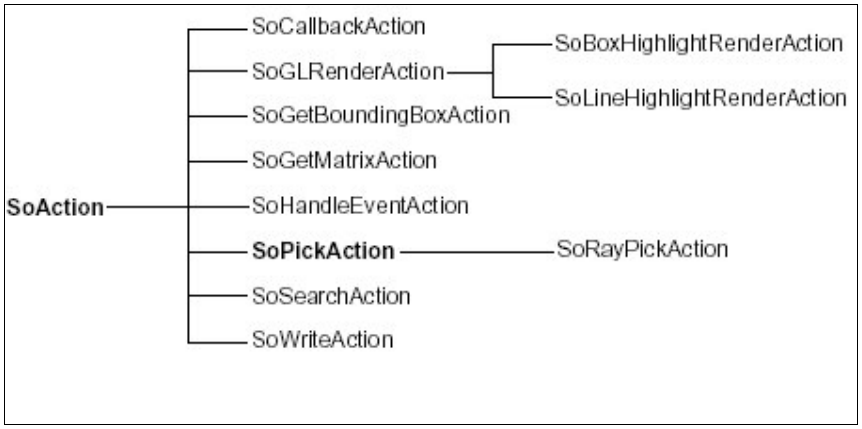


Figura B-7 - Classes que representam as ações

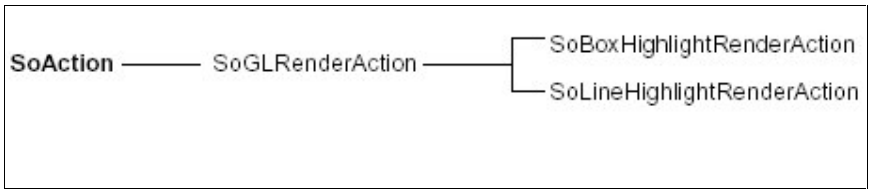


Figura B-8 - Classes que representam destaque

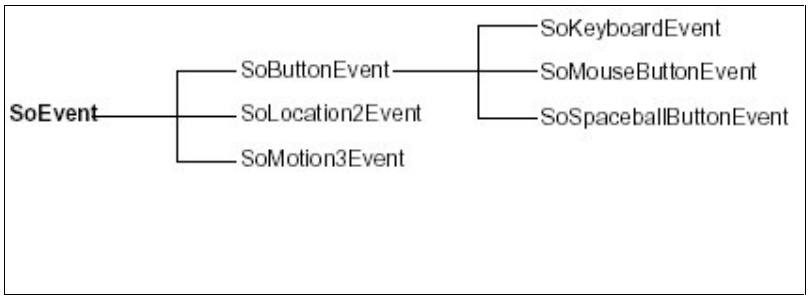


Figura B-9 - Classes que representam eventos

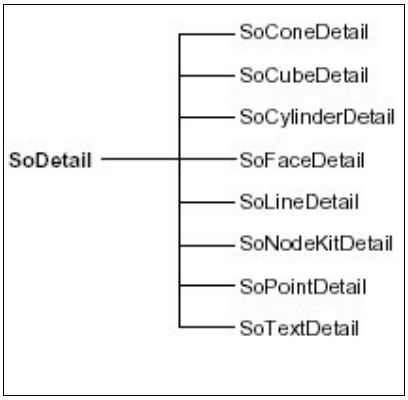


Figura B-10 - Classes que representam detalhes

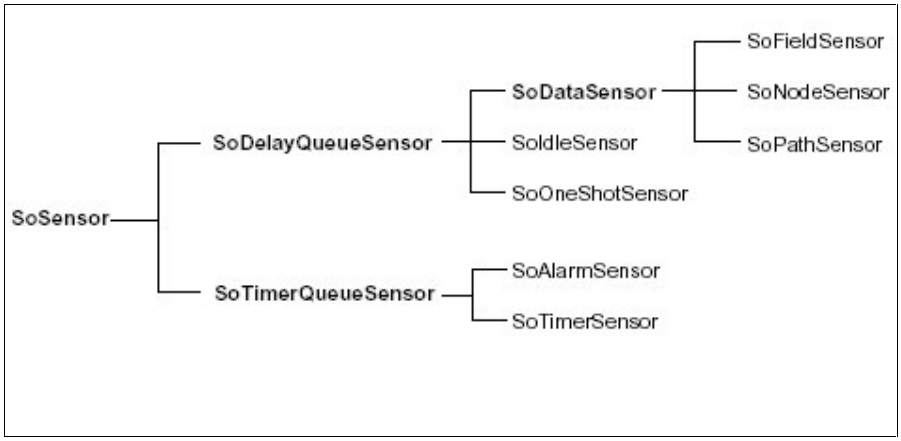


Figura B-11 - Classes que representam sensores

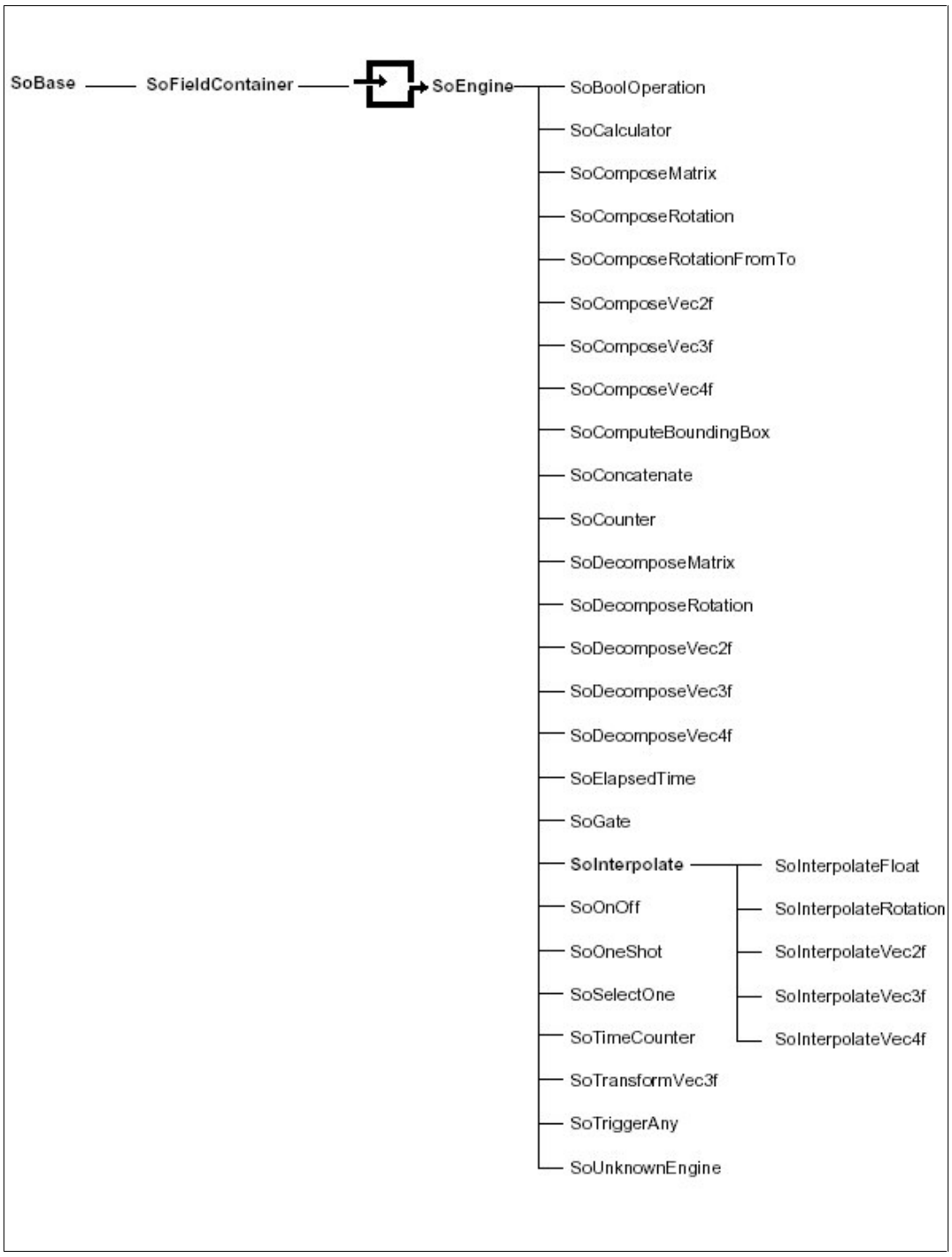


Figura B-12 - Classes que representam *engine*

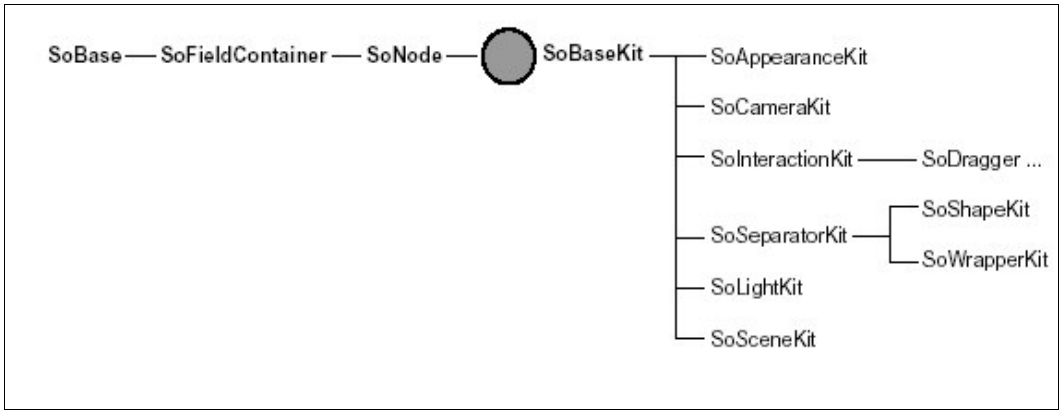


Figura B-13 - Classes que representam kits de nós

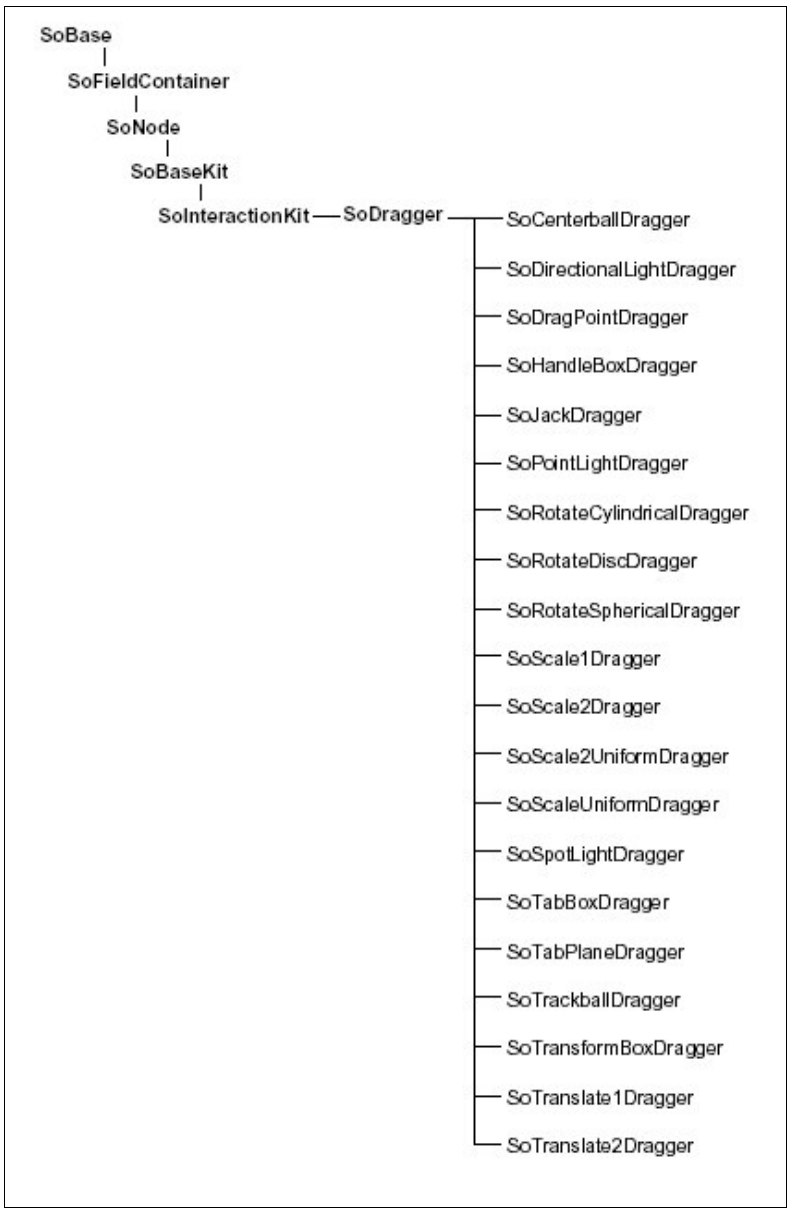


Figura B-14 - Classes que representam *dragger*

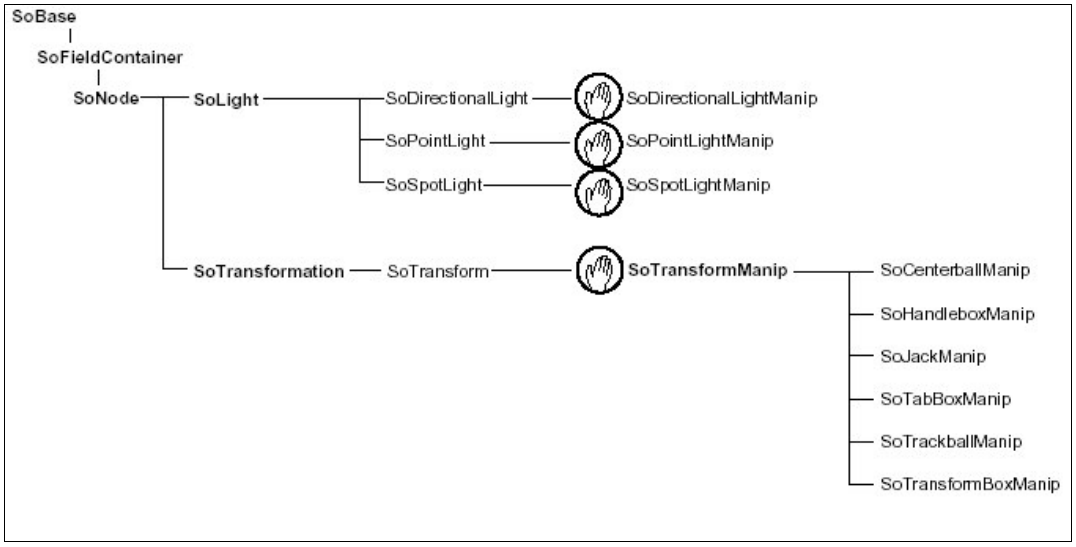


Figura B-15 - Classes que representam manipuladores

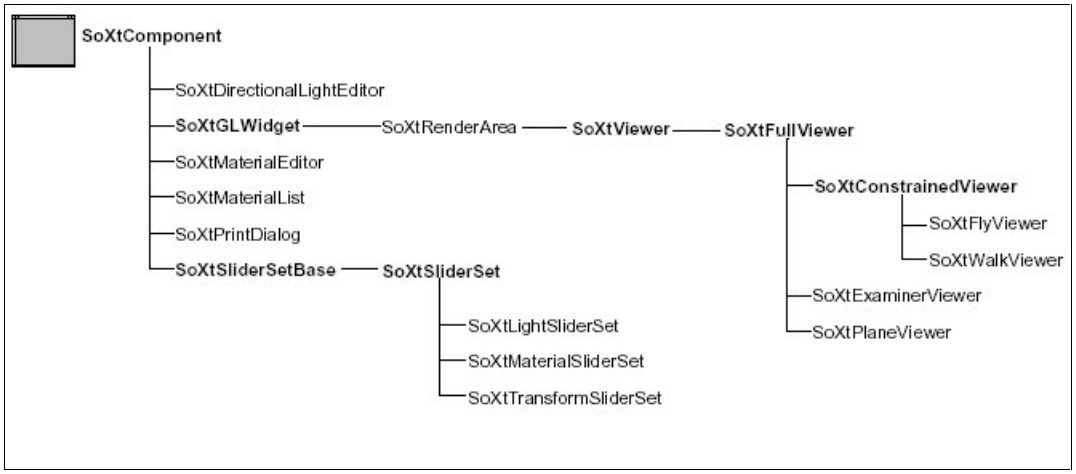


Figura B-16 - Classes que representam os componentes Xt

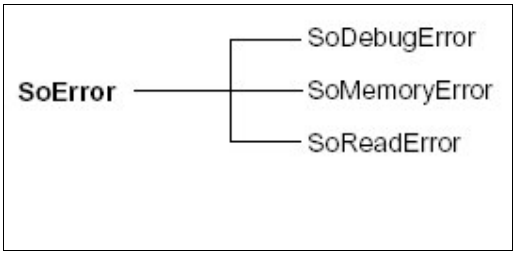


Figura B-17 - Classes que representam os erros

C - Algoritmos de detecção de colisão.

C.1 - Lin-Canny Closest Features Algorithm

Este algoritmo mantém um par de características (vértices, bordas, faces) mais próximas entre a dois poliedros convexos que se movem no espaço. Explorando o fato que as características mais próximas atuais estão provavelmente perto das características mais próximas precedentes, é possível estabelecer um intervalo de tempo constante para realizar a obtenção dos novos valores de tais características. A distância entre dois poliedros convexos é encontrada facilmente uma vez que as características mais próximas são conhecidas. Uma colisão é declarada quando esta distância cai abaixo de algum valor muito pequeno.

C.2 - V-Clip

O algoritmo Voronoi-Clip, ou o V-Clip, segue os pares de características mais próximas de forma similar ao algoritmo de Lin-Canny. Segundo o autor, o V-Clip possuiu uma codificação mais simples e mais robusta que o Lin-Canny. Ele trabalha com poliedros que se interpenetram. Ele também permite a modelagem de poliedros não convexos através do agrupamento de vários poliedros convexos. O V-Clip funciona da seguinte forma: uma região de Voronoi associada com uma caracteristica é o conjunto dos pontos mais próximo dessa característica do que todas as outras características. Todas as regiões de Voronoi de um poliedro criam uma partição do espaço que cerca o poliedro. De acordo com a figura C-1, se um ponto Vb do objeto B se encontrar dentro da região de Voronoi da característica Ea do objeto A, então Ea é a característica mais próxima ao ponto Vb. Similarmente, se o ponto Pa se encontrar dentro da região de Voronoi da característica Vb, então Vb é a característica mais próxima do objeto B para o ponto Pa. Quando estas duas condições são satisfeitas diz-se que o vértice Vb e borda Ea são um par de características mais próximas e a distância entre eles é a menor distância entre o par de poliedros do objeto A e do objeto B. Cada estrutura de dados do poliedro possui campos para suas caracteristicas (faces, vértices, bordas) e regiões de Voronoi. A estrutura de dados para cada característica possui campos para seus parâmetros geométricos e características vizinhas e regiões de Voronoi. Por exemplo, uma borda pode ter um vértice inicial, um vértice final, uma face do lado esquerdo e

outra face do lado direito. A borda também possui sua região de Voronoi que é descrita por quatro planos circunvizinhos. Todas as estruturas de dados são carregadas com dados estáticos durante a inicialização, e as regiões de Voronoi são pré-computadas para cada característica.

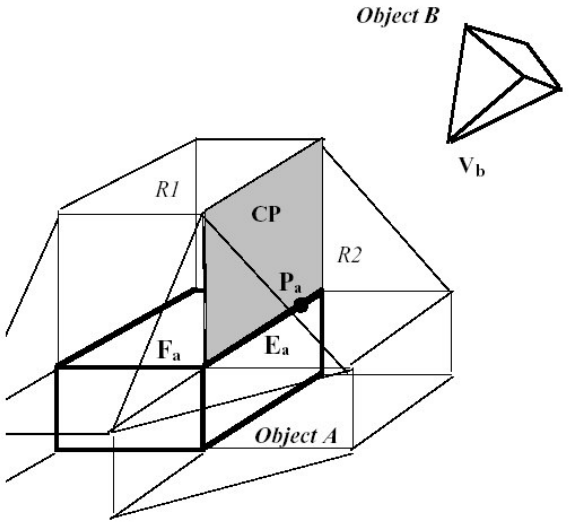


Figura C-1 - R1 e R2 são regiões de Voronoi de Fa e Ea [420]

C.3 - I-COLLIDE

I_COLLIDE é uma biblioteca para detecção de colisão interativa e exata para ambientes grandes compostos de poliedros convexos. Muitos poliedros não convexos podem ser decompostos em grupos de poliedros convexos, que podem então ser usados com esta biblioteca. I_COLLIDE explora a propriedade de uma simulação mudar muito pouco entre etapas consecutivas do tempo e as propriedades dos poliedros serem convexos para conseguir a detecção muito rápida da colisão que é exata para a acuracidade dos modelos de entrada. A biblioteca foi testada em simulações de multi-corpos e em simulações baseadas em impulso. O tempo requerido para a detecção da colisão é tipicamente pequeno comparado ao tempo necessário para gerar os gráficos para estas simulações.

C.4 - OBB-Tree

Uma OBB-Tree é uma representação hierárquica usando caixas delimitadoras orientadas (OBB's - Oriented Bounding Boxes). Uma OBB é uma caixa delimitadora

retangular em uma orientação 3D arbitrária. Idealmente, a OBB é orientada de tal forma que o objeto fique completamente envolvido com o menor volume possível que permita a inclusão total da geometria em questão. Monitora-se a o contato ou colisão do objeto através do contato ou colisão da caixa envoltória delimitadora do volume do objeto.

C.5 - Q-COLLIDE

Q- COLLIDE é um algoritmo simples para detecção exata de colisão para poliedros convexos. O algoritmo encontra rapidamente um plano de separação entre dois poliedros se eles não estiverem colidindo, ou reporta a colisão e os pares de pontos mais próximos entre si se não puder encontrar um plano de separação entre os poliedros. No caso de não haver colisão, o tempo gasto para encontrar o plano de separação entre os poliedros é armazenado para ser utilizado como tempo aproximado para encontrar o próximo plano de separação.

D - Diagrama de seqüência do simulador.

Referências Bibliográficas

[ADADE] ADADE FILHO, A., GÓES, E. **Simulation of the dynamics and control of robotic manipulators using SIMULINK**. Proceedings of the IASTED International Conference on Modelling and Simulation, Pittsburgh, Pennsylvania - USA, 1998. Disponível em http://www.mec.ita.cta.br/~adade/ftp/publications/CBA098.zip. [Capturado em dezembro 2001].

[BARAF] BARAFF, D. **Rigid Body Simulation I – Unconstrained Ridig Body Dynamics**. Robotics Institute Carnegie Mellon University, 1997. Disponível em http://www-2.cs.cmu.edu/~baraff/sigcourse/. [Capturado em novembro de 2001]

[BARENBURG] BARENBRUG, B. G. B. **Designing a Class Library for Interactive Simulation of Rigid Body Dynamics**. Tese de Doutorado. Eindhoven University of Technology. April 2000. Disponível em http://www.win.tue.nl/dynamo/publications/bartbthesis.pdf. [Capturado em maio 2003].

[BLENDER] BLENDER. Disponível em: http://www.blender.org. [Capturado em janeiro de 2003].

[BOURG] BOURG, D. M. **Physics for Game Developers**. O'REILLY, January 2002.

[CAREY] CAREY, R., BELL, G. **The ONLINE Annotated VRML 97 Reference**. Disponível em http://www.web3d.org/resources/vrml_ref_manual/. [Capturado em dezembro de 2001].

[CORKE] CORKE, P. I. **Symbolic Algebra for Manipulator Dynamics**. CSIRO Division of Manufacturing Technology, Australia, 1996.

[COUTINHO] COUTINHO, M. G. **Dynamic Simulations of Multibody Systems**. Springer-Verlar, 2001.

[CRAIG] CRAIG, J.J. **Introduction to Robotics, Mechanics and Control**. Addison-Wesley, 1986.

[DYMOLA] DYMOLA. Disponível em: http://www.dynasim.se/. [Capturado em fevereiro de 2002].

[ECOSIMPRO] ECOSIMPRO. Disponível em: http://www.ecosimpro.com/. [Capturado em dezembro de 2001].

[ELMQVIST] ELMQVIST, H., MATTSSON, S. E., OTTER, M. **Modelica — A Language for Physical System Modeling, Visualization and Interaction**. The 1999 IEEE Symposium on Computer-Aided Control System Design, CACSD'99, Hawaii, 1999. Disponível em: http://www.modelica.org/papers/ModelicaCACSD99.pdf. [Capturado em março de 2002].

[FOLEY] FOLEY, J. D., van DAM, A., FEINER, S. K., HUGHES, J. F. **Computer Graphics principles and practice**. Addison Wesley, 1996.

[GAMMA] GAMMA, E., HELM R., JOHNSON R., VLISSIDES, J. **Design Patterns - Elements of Reusable Object-oriented Software**. Addison-Wesley Professional Computing Series, 1995.

[GAO] GAO, J. **Collision Detection**. Disponível em http://www.stanford.edu/~jgao/collision-detection.html. [Capturado em janeiro 2002].

[GILLESPIE] GILLESPIE, R. B., COLGATE, J. E. **A Survey of Multibody Dynamics for Virtual Enviroments**. Department of Mechanical Engineering Northwestern University, 19XX.

[GOLDSTEIN] GOLDSTEIN, H. **Classical Mechanics**. Addison Wesley, 1972.

[HECKER] HECKER, C. Communications of the ACM – July 2000/Vol. 43, Nº7.

[ISSO/IEC 14772] ISO/IEC 14772-1:1997. **Information technology – Computer graphics and image processing – The Virtual Reality Modeling Language (VRML) – Part1: Functional specification and UTF-8 encoding**, 1997. Disponível em http://www.web3d.org/technicalinfo/specifications/ISSO_IEC_14772-All/index.html. [Capturado em janeiro de 2001].

[KELLER] KELLER, H., STOLZ H., ZIEGLER, A., BRAUNL, T. **AERO - A Physically Based Simulation and Animation System**. Disponível em www.aero-simulation.de. [Capturado em agosto 2002].

[KOH] KOH, A., **freeCAD – 3D CAD with Motion Simulation**. Disponível em: http://www.askoh.com/freecad/index.html. [Capturado em outubro de 2002].

[MOORE] MOORE, M., WILHELMS, J. **Collision detection and response for Computer Animation**. ACM Computer Graphics, volume 22, number 4, August 1988.

[MORFIT] MORFIT. Disponível em: http://www.morfit.com. [Capturado em março de 2001].

[MSC ADAMS] MSC ADAMS. Disponível em: http://www.adams.com. [Capturado em dezembro de 2001].

[NASA] NASA Space Telerobotics Program. **Photograph Archive**. Disponível em http://ranier.hq.nasa.gov/telerobotics_page/photos.html. [Capturado em outubro 2002].

[NEWPORT] Newport Corporation. **Tutorials: Motion Control**. Disponível em http://www.newport.com/Support/Tutorials/Motion_Control/. [Capturado em janeiro de 2002].

[NR in C] **Numerical Recipes in C: The Art of Scientific Computing**, Cambridge University Press, 1992. Disponível em http://www.nr.com. [Capturado em janeiro de 2002].

[OPEN INVENTOR] Open Inventor Architecture Group. **Open Inventor C++ Reference Manual – The Official Reference Document For Open Inventor, Release 2.** Addison Wesley, 1994.

[OPENFX] OPENFX. Disponível em: http://www.openfx.org. [Capturado em abril de 2001].

[OPENGL] OPENGL. Disponível em: http://www.opengl.org. [Capturado em setembro de 2001].

[PAZOS] PAZOS, A. F. **Robôs manipuladores – 2ª. Parte**. Mecatrônica Atual no. 3, Abril 2002.

[QT] Trolltech. Disponível em http://www.trolltech.com. [Capturado em julho de 2003].

[RADOK] RADOK, J. R. M., **Treatise of Physics in Elementary Presentation - General Principles of Mechanics**. Disponível em http://kr.cs.ait.ac.th/~radok/physics/C1.htm#C1. [Capturado em agosto 2002].

[ROSSUM] ROSSUM, G. van. **Extending and Embedding the Phyton Interpreter Release 2.2**. PhytonLabs, 2001. Disponível em http://www.python.org/doc/current/ext/ext.html. [Capturado em janeiro 2002].

[SHREINER] SHREINER, D. **OpenGL Reference Manual: the official reference document to OpenGL, version 1.2**. Addison-Wesley, 2000.

[SILICON GRAPHICS] Silicon Graphics. **Open Inventor General FAQ**. Disponível em http://oss.sgi.com/projects/inventor/genfaq.html. [Capturado em setembro 2001].

[SILVA F.] SILVA, F. W. S. V., **Um sistema de animação baseado em movimento capturado**. Dissertação de mestrado COPPE/UFRJ, 1998. Disponível em http://w3.impa.br/~nando/publ/thesis-letter.pdf. [Capturado em maio de 2003].

[SILVA M.] SILVA, M. H. **Dissertação de Mestrado: "TOOLKIMA: Uma ferramenta para Animação Modelada por Comuputador"**. DCA – FEE – UNICAMP – Abril – 1992.

[SPONG] SPONG, M. W., VIDYASAGAR, M. **Robot Dynamics and Control**. 1989.

[SUN] SUN MICROSYSTEMS. **The Java 3D API**. Technical White Paper, 1997. Disponível em http://java.sun.com/products/java-media/3D/collateral/j3d_api/j3d_wp_SMCC.pdf. [Capturado em janeiro de 2001].

[VRML] VRML. Disponível em: http://www.vrml.org. [Capturado em dezembro de 2001].

[VRMLPAD] VrmlPad. Disponível em http://www.parallelgraphics.com/products/vrmlpad. [Capturado em maio de 2003].

[WABIAN] WABIAN (WAseda BIpedal humANoid). Humanoid Project, Advanced Research Institute for Science and Engineering, Waseda University. Disponível em http://www.fzi.de/divisions/ipt/WMC/preface/node330.html. [Capturado em outubro 2002].

[WALSH] WALSH, A. E. **Using graph-based data structures to organize and manage scene contents.** Dr. Dobb's Journal, July 2002. Disponível em http://www.ddj.com/documents/s=7217/ddj0207a/0207a.htm. Capturado em agosto de 2002.

[WERNECKE] WERNECKE, J. **The Inventor Mentor Programming Object-Oriented 3D Graphics with Open InventorTM, Release 2**. Addison Wesley, 1994.

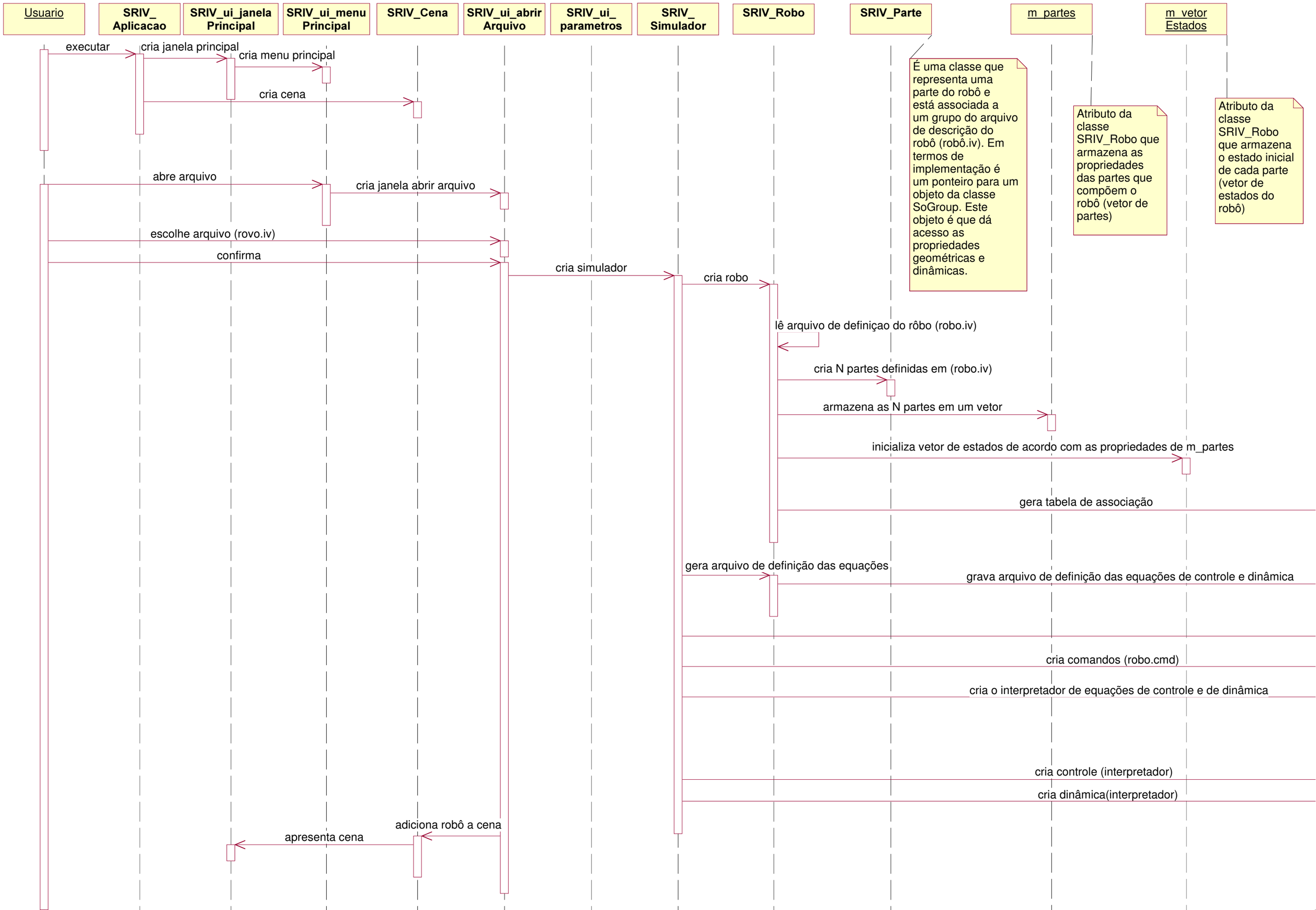
[WITKIN 1] WITKIN, A. **An Introducton to Physically Based Modeling: Contrained Dynamics**. Robotics Institute Carnegie Mellon University, 1997.

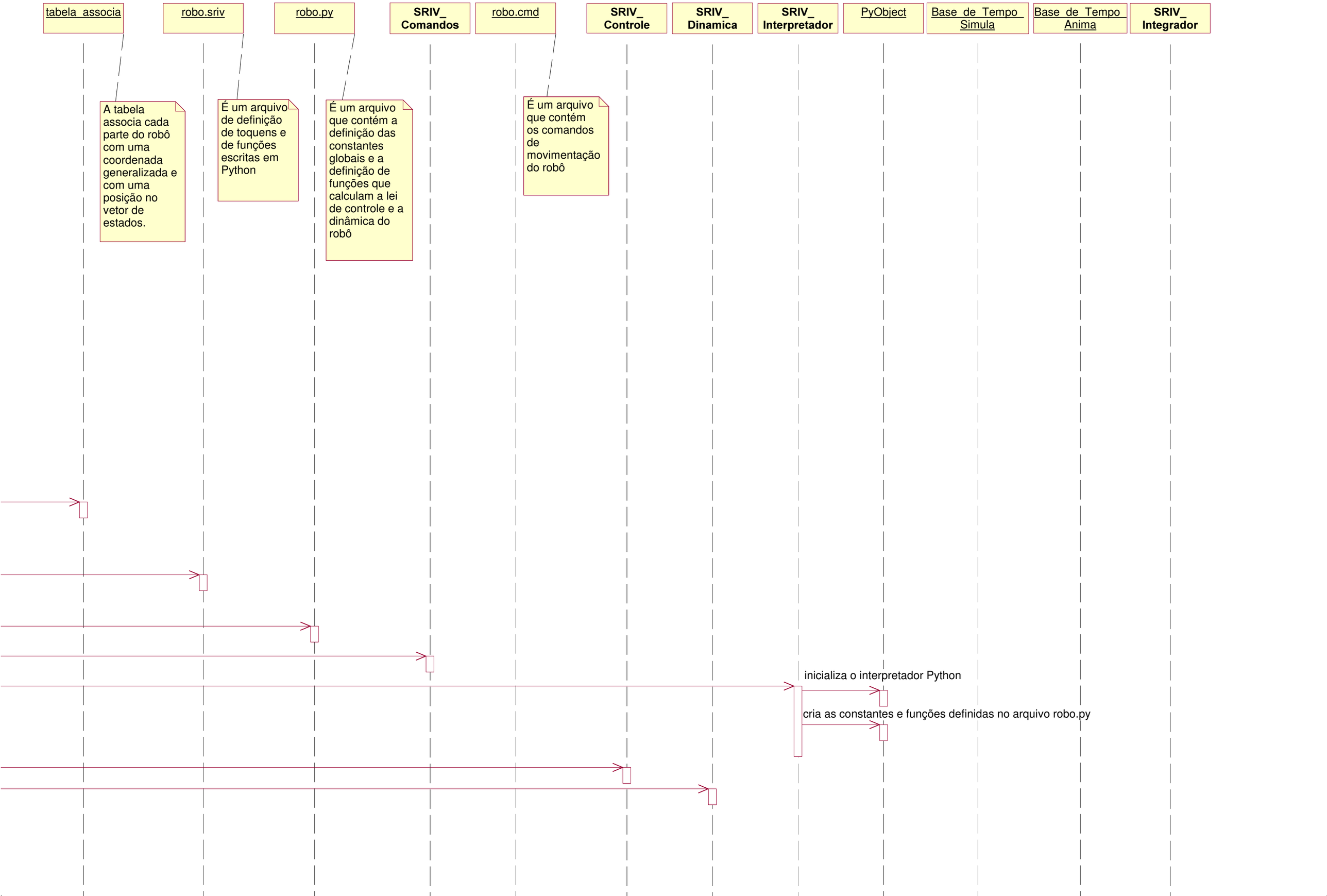
[WITKIN 2] WITKIN, A. **Particle system dynamics**. Robotics Institute Carnegie Mellon University, 1997.

[WITKIN 3] WITKIN, A. **Physically Based Modeling: Principles and Practice Particle System Dynamics**. Robotics Institute Carnegie Mellon University, 1997.

[ZHANG] ZHANG, P. **Phisically Realistic Simulation of Rigid Bodies**, Dissertação de Mestrado em Ciência da Computação na Escola de Engenharia de da Universidade de Tulane, 1996. [420]

D - Diagrama de seqüência do simulador.




tabela_associa
robo.sriv
robo.py
SRIV_ Comandos
robo.cmd
SRIV_ Controle
SRIV_ Dinamica
SRIV_ Interpretador
PyObject
Base_de_Tempo Simula
Base_de_Tempo Anima
SRIV_ Integrador
A tabela associa cada parte do robô com uma coordenada generalizada e com uma posição no vetor de estados.
É um arquivo de definição de toquens e de funções escritas em Python
É um arquivo que contém a definição das constantes globais e a definição de funções que calculam a lei de controle e a dinâmica do robô
É um arquivo que contém os comandos de movimentação do robô
inicializa o interpretador Python
cria as constantes e funções definidas no arquivo robo.py

define parâmetros

abre janela de definição de parâmetros

atualiza parâmetros

valida parâmetros

confirma

inicia simulação

Inicialização da Simulação

inicia simulação

cria relógio de sincronismo (ExecutaSimulacao( ))

insere relógio da simulação na fila de eventos da aplicação

cria relógio de sincronismo (Anima( ))

insere relógio da animação na fila de eventos da aplicação

executa simulação

simula

recupera vetor de estados do robô

cria vetor de estado atual com os valores de (m_vetorEstado, número de estados)

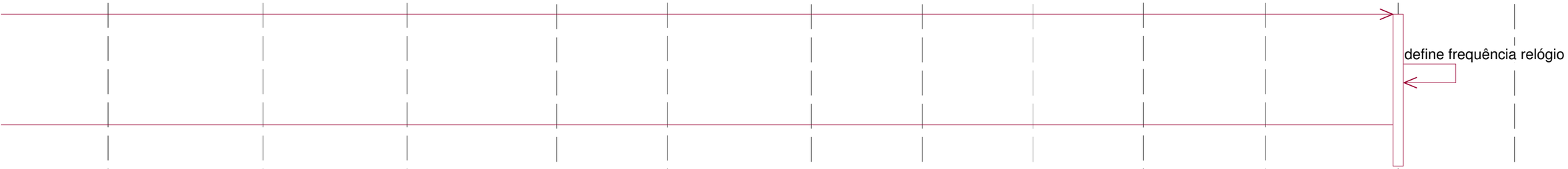
cria vetor de estado desejado vazio (número de estados)

cria vetor de estado novo vazio (número de estados)

cria o vetor de atuação vazio()

Executa Ciclo


define frequência do relógio


define frequência relógio

Tratando os Eventos de Executa Simulação e Executa

executa comandos (tempo corrente)

rotorna valores para (vetor de estados desejados)

gera atuação (vetor de estado atual, vetor de estado desejado)

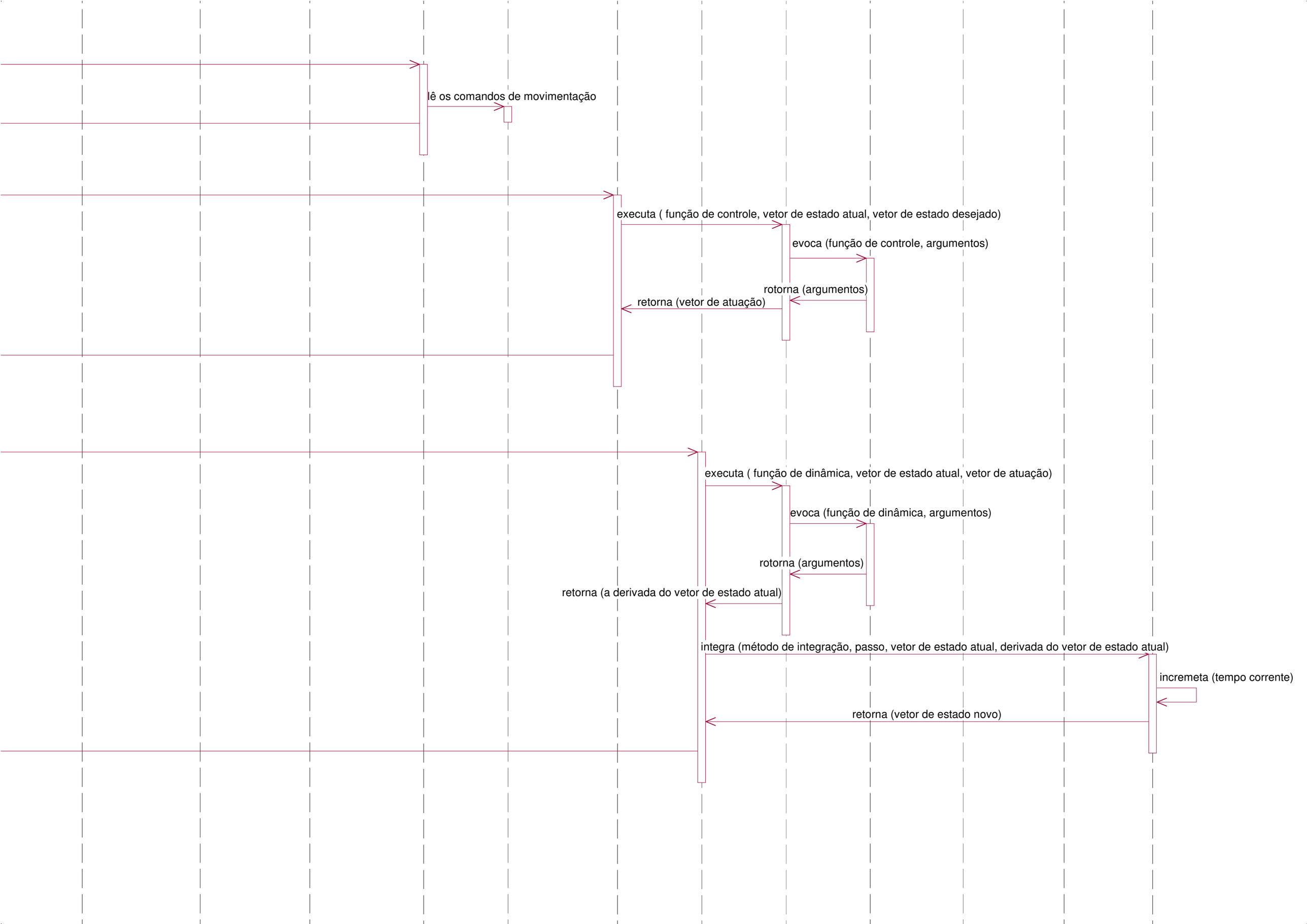
retorna valores para (vetor de atuacao)

calcula dinâmica (vetor de estado atual, vetor de atuação)

retorna valores para (vetor de estado novo)

atualiza vetor de estados do robô (vetor de estado novo)

atualiza


lê os comandos de movimentação
executa (função de controle, vetor de estado atual, vetor de estado desejado)
evoca (função de controle, argumentos)
rotorna (argumentos)
retorna (vetor de atuação)
executa (função de dinâmica, vetor de estado atual, vetor de atuação)
evoca (função de dinâmica, argumentos)
rotorna (argumentos)
retorna (a derivada do vetor de estado atual)
integra (método de integração, passo, vetor de estado atual, derivada do vetor de estado atual)
incremeta (tempo corrente)
retorna (vetor de estado novo)

Animação

atualiza posição e orientação das partes do robô

anima

varre tabela de associação para saber quais partes do robô estão relacionadas com o vetor de estado do rob

atualiza posição e orientação de cada parte associada ao vetor de estado do robô

para simulação

encerra simulação

destrói relógio de sincronismo da simulação

destrói relógio de sincronismo da animação

retira robô da cena