RAFAEL COUTO BUENO

Modelagem e co-simulação de sistemas mecatrônicos híbridos

São Paulo

2013

RAFAEL COUTO BUENO

Modelagem e co-simulação de sistemas mecatrônicos híbridos

Trabalho de Conclusão de Curso apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do título de Bacharel em Engenharia.

São Paulo

2013

RAFAEL COUTO BUENO

Modelagem e co-simulação de sistemas mecatrônicos híbridos

Trabalho de Conclusão de Curso apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do título de Bacharel em Engenharia.

Área de Concentração: Engenharia Mecatrônica

Orientador: Prof. Dr. Fabrício Junqueira

São Paulo

2013

FICHA CATALOGRÁFICA

Bueno, Rafael Couto
Modelagem e co-simulação de sistemas mecatrônicos híbridos / R.C. Bueno. -- São Paulo, 2013.
72 p.

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos.

1. Sistemas híbridos 2. Sistemas embutidos 3. Lógica modal 4. Mecatrônica (Sistemas) I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos II. t.

Dedico este trabalho aos meus pais Ricardo e Claudia, ao meu irmão Gabriel e minha irmã Victória, pelos momentos de apoio e compreensão, principalmente neste último ano.

AGRADECIMENTOS

Ao **Prof. Fabrício Junqueira**, pelo interesse, paciência, compreensão e dedicação durante as orientações e elaboração deste trabalho, pelo apoio em momentos difíceis, minha sincera admiração e gratidão.

Ao **Prof. Matteo Rossi,** orientador da parte inicial do trabalho realizado no Politecnico di Milano na Itália, pela sugestão do tema, todo o auxilio e compreensão quanto à escrita em italiano do trabalho e por toda a igual dedicação durante as orientações e elaboração da parte inicial deste trabalho.

Ao **Prof. Lucas Antonio Moscato, Prof. Thiago de Castro Martins, Profª Larissa Driemeier** e **Prof. Arturo Forner Cordero**, pelo apoio e compreensão durante a fase de qualificação do trabalho.

Aos **nossos Mestres** do curso de graduação em Engenharia Mecatrônica, por todo o conhecimento, experiência e dedicação ao longo do curso.

À **Escola Politécnica da Universidade de São Paulo**, por todo o conhecimento adquirido ao longo de minha formação.

Ao **Politecnico di Milano**, pelo conhecimento adquirido nesta etapa da minha formação.

RESUMO

Em sistemas onde existe uma parte que trabalha no continuo e outra que trabalha no discreto (como sistemas mecatrônicos embarcados) surge o problema da dificuldade de unificar os modelos das partes continua e discreta em um único modelo. A solução proposta é o uso da co-simulação. Uma das formas de fazer isto é por meio de um software desenvolvido num projeto europeu para projeto e simulação de sistemas embarcados, o MCA (*MADES Co-simulation Approach*). Tal software permite o projeto do sistema de controle por meio de regras lógicas, que podem ser usadas pra guiar o funcionamento do sistema. Desta forma prevê-se o uso destas regras pra garantir a segurança do sistema. O caso escolhido para estudo foi de dois braços robóticos executando tarefas no mesmo espaço de trabalho, garantindo que eles não colidam entre si nem com possíveis intrusos. Espera-se com este projeto avaliar a validade do uso de tal programa para projeto de softwares embarcados, avaliando o tempo de execução da simulação, e a facilidade de aprendizagem do método.

Palavras-chave: Co-simulação, Sistemas híbridos, MADES, Sistemas mecatrônicos, Sistemas embarcados, Lógica temporal, Segurança, Continuo/Discreto.

ABSTRACT

In systems where there is a part that works in continuous and one that works in discrete (like mechatronic embedded systems) exists the problem to unify the models of continuous and discrete parts in a single model. The proposed solution is the use of co-simulation, using a software developed in a European project for the design and the simulation of embedded systems, the MCA (*MADES Co-simulation Approach*). This software allows the design of control system by means of logical rules, which can be used to guide the operation of the system. Thus it provides the use of these rules to ensure the safety of the system. The study case chose was two robotic arms performing tasks in the same workplace, ensuring that they do not conflict with each other or with possible intruders. It is hoped that this project to evaluate the validity of using such program to project embedded software systems, evaluating the runtime of the simulation, the ease of learning the method.

Keywords: Co-simulation, Hybrid systems, MADES, Mechatronic systems, Embedded systems, Temporal logic, Security, Continuous/Discrete.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1. INTRODUÇÃO

O projeto de um sistema embarcado exige uma análise cuidadosa de todos os elementos envolvidos (BARESI et al., 2012), pois à medida que os sistemas estão evoluindo, estão tornando-se cada vez mais complexos, levando inevitavelmente à presença de uma maior quantidade de erros não previstos no projeto (BAGNATO et al., 2010).

Torna-se relevante à implantação de tais sistemas o uso de metodologias e ferramentas, que por meio de modelos virtuais, auxiliem os desenvolvedores desde o projeto até a verificação (teste) e validação do sistema (BAGNATO et al., 2010).

Esta prática mostra-se essencial quando, por exemplo, alguma parte envolvida no processo (mecanismo, alvo ou até mesmo o ambiente ao redor) pode sofrer danos se for operado fora da sua zona de operação segura, ou quando protótipos ainda não estão disponíveis para testes, ou ainda quando nem é possível construir protótipos de um dado sistema ou processo (BROENINK et al., 2008).

Na área mecatrônica é frequente deparar-se com situações deste tipo, como por exemplo em sistemas embarcados em aeronaves (BAGNATO et al., 2010), em satélites (BARESI et al., 2012), ou ainda em sistemas autônomos, com robôs cooperando e colaborando para a realização de uma tarefa (BASILE et al., 2013).

Outro fator é que em sistemas de manufatura e de automação industrial, as restrições temporais e a presença de interações complexas complicam ainda mais o projeto do sistema, evidenciando a necessidade do uso de ferramentas de auxílio (BARESI et al., 2012).

Esta prática, independentemente da metodologia escolhida, implica na modelagem completa do dado sistema em um ambiente virtual, incluindo também a modelagem do ambiente onde o mesmo estará inserido, a fim de poder realizar simulações de seu funcionamento nas condições onde o sistema será utilizado (BROENINK et al., 2008) e (BAGNATO et al., 2010).

A dificuldade intrínseca para realizar esta modelagem quando aplicada a um sistema embarcado é que dentro deste modelo coexistirão dois subsistemas de natureza completamente diferente: a parte física do sistema, que engloba o sistema dinâmico controlado e o ambiente onde está inserido, que é expresso em tempo contínuo, e o próprio sistema embarcado que é expresso em tempo discreto (BARESI et al., 2012). Isto é, um sistema híbrido, que pode ser definido como um sistema dinâmico, cujo comportamento exibe tanto as alterações discretas

como contínuas, e tipicamente é constituído por uma coleção de programas digitais, que interagem uns com os outros e com um ambiente analógico (HENZINGER et al., 1997).

Os sistemas híbridos incluem desde controladores de produção e aos já referidos sistemas embarcados, como controladores de voo e automotivos, equipamentos médicos, sistemas micro- eletromecânicos, e robôs (HENZINGER et al., 1997).

Estes sistemas criam um problema exatamente para a integração destes dois subsistemas numa única solução (BARESI et al., 2012), por meio de uma única técnica e/ou formalismo, demandando complexas derivações analíticas, como os autômatos híbridos (HENZINGER et al., 1997).

Outro fator é que, devido exatamente a esta natureza intrínseca diferente, contínuo versus discreto, a maioria das abordagens para o projeto destes é baseado em diferentes modelos e ferramentas de desenvolvimento tanto para o ambiente, comumente chamado o modelo físico, como para o sistema, como é chamado o modelo de *software* (BARESI et al., 2012).

Neste ponto, para conseguir unificar estes modelos surge a ideia de criar programas responsáveis por criar essa ponte entre o ambiente (expresso em tempo continuo) e o sistema (expresso em tempo discreto). Esses programas são chamados co-simuladores e conseguem também avaliar o comportamento coordenado dos dois subsistemas, evitando possíveis erros (AL-HAMMOURI et al., 2008).

Portanto a co-simulação permite o uso de ferramentas de simulação diferentes para cada parte (continuo e discreto), ou seja, cria-se e simula-se cada modelo independentemente, em módulos separados, que depois são interconectados pelo co-simulador para realizar as simulações simultaneamente, de forma paralela, muitas vezes sendo capazes de trocar informações entre si (BARESI et al., 2012), de forma colaborativa, conforme Fig. 1.

A Fig. 1 evidencia os aspectos importantes de um co-simulador, a simulação independente de cada subsistema, com variáveis compartilhadas por meio do coordenador (Λ_A e Λ_S) e comandos do mesmo para guiar as simulações independentes de modo a obter um resultado único da simulação combinada dos dois subsistemas. Em uma visão mais ampla, o co-simulador utiliza o modelo do sistema e do ambiente, a fim de construir a evolução do sistema de maneira que esta seja concebível por ambos os modelos, isto é, que não viola quaisquer condições em cada um deles.

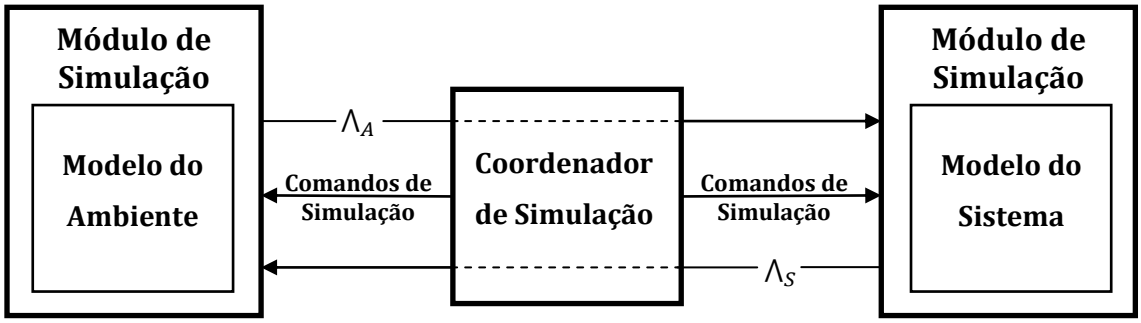

Figura 1.1 – Representação esquemática da arquitetura de um co-simulador – baseada na esquematização feita por Baresi et al. (2012).

Neste trabalho será utilizada a metodologia de co-simulação e o co-simulador associado estabelecidos em Baresi et al. (2012), o MCA (*MADES Cossimulator Approach*), que estabelece a modelagem do ambiente (parte contínua) na linguagem Modelica (FRITZSON, 2004) e a modelagem do sistema (parte discreta) por meio de regras lógicas temporais definidas dentro da ferramenta Zot (PRADELLA, 2009).

Esta metodologia possui a vantagem de ser não determinística, ou seja, desde que continuem sendo respeitadas as regras definidas, o co-simulador tem possibilidade de escolher, segundo uma lógica própria, qual a melhor opção para o andamento do sistema, dentro do intervalo de opções possíveis para a evolução do sistema que respeitam essas regras.

1.1. Objetivos do trabalho

Os sistemas embarcados são muitos usados na área mecatrônica (BAGNATO et al., 2010, BARESI et al., 2012 e BASILE et al., 2013), e a técnica da co-simulação fornece uma ferramenta alternativa ao projeto destes.

Assim o objetivo deste trabalho será compreender o uso desta técnica, e por meio da escolha de um co-simulador, estabelecer uma metodologia para aplicá-lo a qualquer sistema mecatrônico embarcado e estudar a sua aplicação por meio de um estudo de caso.

Esse estudo de caso tem como objetivo avaliar a viabilidade do uso desta metodologia na prática, identificando as vantagens e desvantagens desta abordagem e, se necessário, salientar as partes do programa nas quais se vê necessário fazer melhorias.

Busca-se também estabelecer quais são os pré-requisitos e a dificuldade de aprendizado básico necessário ao uso do programa, como por exemplo, para programar com as regras lógicas.

1.2. Estrutura do trabalho

Este trabalho está dividido em sete capítulos. No capítulo 2 (desenvolvimento) é apresentado o estado da arte, o funcionamento dos programas e linguagens utilizados e os conceitos teóricos necessários à compreensão do trabalho.

No capítulo 3 tem-se o estabelecimento da metodologia necessária ao uso do co-simulador e de como é feita a modelagem de um sistema genérico com este programa. No capítulo 4 tem-se o estudo de caso, com os resultados apresentados no capítulo 5, e por fim a conclusão no capítulo 6.

Para o estudo de caso foi utilizado um sistema mecatrônico simples, uma série de braços robóticos trabalhando no mesmo espaço operativo, de modo a demonstrar a modelagem, aproximação, a verificação da segurança, a análise paramétrica, e a utilização da ferramenta.

O foco deste caso de estudo foram os requisitos de segurança, com o objetivo de evitar que o braço robótico colida com uma pessoa, objeto ou até mesmo com outro braço que trabalha no mesmo espaço.

2. REVISÃO BIBLIOGRÁFICA

Faz-se necessária uma breve análise dos trabalhos mais relevantes da área, incluindo a fundamentação teórica básica para o entendimento do uso da co-simulação. A teoria abordada neste trabalho não objetiva demonstrações ou interpretações das complexas equações envolvidas; para tanto, recomenda-se a leitura das referências indicadas.

Esta seção está dividida conforme a cronologia dos estudos dos métodos de co-simulação. Primeiramente são apresentadas as metodologias já propostas, comparando com a metodologia MCA escolhida e, posteriormente, explicando o funcionamento desta. Por fim são explicadas as bases teóricas dos métodos usados para a modelagem do ambiente e do sistema, a modelagem não causal e as lógicas temporais respectivamente.

2.1. Metodologias de co-simulação

A co-simulação vem sendo amplamente estudada nos últimos anos e foram já propostas várias soluções, cada uma com suas qualidades e limitações. Por exemplo, o Stateflow, desenvolvido pela MathWorks (2003), é uma ferramenta lógica de controle utilizada para modelar os sistemas híbridos reativos (HAREL; PNUELI, 1985) por meio de diagramas de estado e fluxo dentro de um modelo Simulink.

Segundo a MathWorks (2003), ele usa uma variante da notação da máquina de estado finito criada por Harel (1987), e usando um programa complementar para geração de códigos, pode-se gerar automaticamente códigos, por exemplo, em C.

Entretanto, o Simulink usa uma abordagem causal para a modelagem, onde submodelos são conectados por portas de entrada e saída, o que pode ser considerado uma desvantagem na modelagem de problemas mais complexos, como justifica Zupancic et al. (2008) e será mais bem detalhada na seção 2.3.

Com o intuito de fornecer uma abordagem menos causal, foi desenvolvido o AMESim (JERE, 2011) que se baseia na ideia de que as variáveis, compartilhadas nas portas entre os submodelos, são entidades físicas e portanto operam em ambas as direções. Para isto utiliza da abordagem teórica dos grafos de ligação (KARNOPP; MARGOLIS; ROSENBERG, 1990; PAYNTER, 1961) como solução para criar os modelos, facilitando a ligação entre os diferentes domínios físicos.

Partindo de uma abordagem totalmente diferente, existem as bibliotecas de co-simulação para o Modelica, linguagem desenvolvida com o objetivo de usar a abordagem não causal para modelagem de sistemas (FRITZSON, 2004).

A biblioteca padrão StateGraph (OTTER; ARZEN; DRESSLER, 2005) e sua atualização StateGraph2 (OTTER et al., 2008) fornecem componentes para modelar tanto eventos discretos como sistemas reativos utilizando da linguagem Modelica. Elas baseiam-se no método JGraphChart (ARZEN; OLSSON; AKESSON, 2002), que une os conceitos do Grafcet (DAVID; ALLA, 1992) com elementos adicionais dos diagramas de estado (HAREL, 1987).

Portanto, estas bibliotecas têm o potencial semelhante à modelagem com diagramas de estado, evitando algumas das suas deficiências, como a modelagem causal. Nota-se, porém, que o modelo é sempre determinístico devido à regra de atribuição única do Modelica.

Outra opção é o ModelicaDEVS (BELTRAME, 2006), uma biblioteca livre desenvolvida no ambiente Modelica, para modelar sistemas orientados a eventos discretos utilizando a metodologia DEVS (ZEIGLER; PRAEHOFER; KIM, 2000).

No entanto, esta metodologia adota uma abordagem completamente diferente para a integração numérica do modelo, desenvolvida por Zeigler e Lee (1998): dado o fato de que todas as simulações em computador se submetem a uma discretização, a ideia básica desta abordagem de integração é a substituição da discretização de tempo por uma discretização de Estado, de tal forma que o sistema não evolui de passo em passo, mas sim de estado para estado. Isso faz com que as variáveis de estado evoluem individualmente, sem a necessidade de atualizá-las simultaneamente.

Como já foi dito, a principal diferença entre a abordagem MCA e as outras ferramentas de simulação que trabalham com sistemas híbridos é a descrição dos modelos de maneira não determinística, o que implica a necessidade de um mecanismo que permita a exploração de diferentes alternativas para a simulação da dupla discreto/contínuo, em caso de violação de uma restrição. A MCA também permite, devido à sua arquitetura, uma verificação formal do sistema de *software*, o que também não é considerada nas abordagens apresentadas, baseadas em conceitos mais tradicionais (BARESI et al., 2012).

2.2. O funcionamento da abordagem MCA e seu co-simulador

A metodologia MCA (BARESI et al., 2012) foi escolhida por permitir ao projetista combinar diferentes formalismos complementares (equações diferenciais e fórmulas lógicas, no caso) de forma contínua, em vez de exigir aos submodelos satisfazer um único modelo com notações únicas, aproveitando-se dos pontos fortes de cada formalismo.

O funcionamento do MCA é baseado no uso do co-simulador MADES (BAGNATO et al., 2010), com a concepção do sistema embarcado por meio de um procedimento desenvolvido por Baresi et al. (2010, 2012).

Este procedimento define que o sistema embarcado é concebido por meio de predicados e axiomas escritos em TRIO (CIAPESSONI et al., 1999), que é uma lógica linear temporal de primeira ordem, e depois verificados quanto à coerência por Zot (PRADELLA, 2009), um programa, desenvolvido no Politecnico di Milano, que verifica a coerência de regras lógicas e cria um modelo de ligação entre elas. Do outro lado, o ambiente é modelado de maneira não causal por meio da linguagem Modelica, por meio da plataforma OpenModelica.

A premissa do MCA é que os modelos do sistema e do ambiente se comunicam por meio de variáveis compartilhadas, possuindo também variáveis privadas que não são visíveis ao outro, como visto na Fig. 2.1.

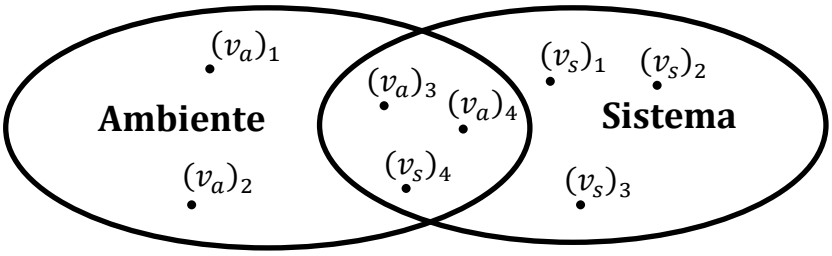

Figura 2.1 – Esquema das variáveis numa co-simulação, baseado em Baresi et al. (2012).

Cada variável pertence a um modelo, por exemplo, $(v_a)_3$ é acessível a ambos os modelos, mas pertence ao modelo do ambiente. Estas variáveis também podem ter valores reais ou sinais discretos finitos, como variáveis lógicas.

A dinâmica das variáveis de ambiente é regulada por equações algébricas ou diferenciais definidas pela linguagem Modelica enquanto as variáveis do sistema estão sujeitos às restrições lógicas definidas pela sintaxe TRIO, que pode impor restrições inclusive às variáveis compartilhadas. As primeiras têm uma noção de tempo contínuo e as outras de tempo discreto.

As duas noções de tempo (contínuo para o modelo de ambiente, discreto para o modelo do sistema) são integradas por meio da noção de amostragem (FURIA; ROSSI, 2010). Se o "período de amostragem" é δ, cada u.t. (unidade de tempo) discreto k corresponde a um instante de tempo contínuo de $k\delta$.

A arquitetura da MCA, seguindo a definição de co-simulação da Fig. 1.1, é resumida da seguinte maneira:

- O modelo do ambiente é escrito na linguagem Modelica;
- O modelo do sistema é escrito por meio de regras lógicas TRIO;
- O módulo de simulação para o ambiente é baseado no intérprete OpenModelica;
- O módulo de simulação para o sistema é baseado na ferramenta Zot, que também atua como um verificador da coerência das regras lógicas temporais; e
- O coordenador, uma aplicação Java *stand-alone*, é uma versão modificada do MADES original, que toma como entrada os modelos de ambiente e de sistema, entre outros parâmetros (ex, intervalo δ de amostragem, tempo total de simulação, etc.), e depois executa um algoritmo descrito por Baresi et al. (2012) de modo a simular o sistema conjunto.

A Fig. 2.2 mostra a interface do co-simulador resultante, onde nota-se que para iniciar a simulação é necessária a especificação de um arquivo no formato XML (*e**X**tensible **M**arkup **L**anguage*), arquivo este que define a relação de variáveis, com o tipo, a origem, e se são privadas ou compartilhadas.

2.3. Abordagem não causal: a linguagem Modelica e o programa OpenModelica

Muitas ferramentas de simulação, como o SIMULINK, utilizam uma abordagem causal, supondo que um sistema pode ser decomposto em estruturas de diagrama de blocos com as interações causais. Para isso as equações devem ser escritas na forma de EDO's (equações diferenciais ordinárias):

$$y^{(n)} = F(y^{(n-1)}, y^{(n-2)}, \dots, y', y, x)$$

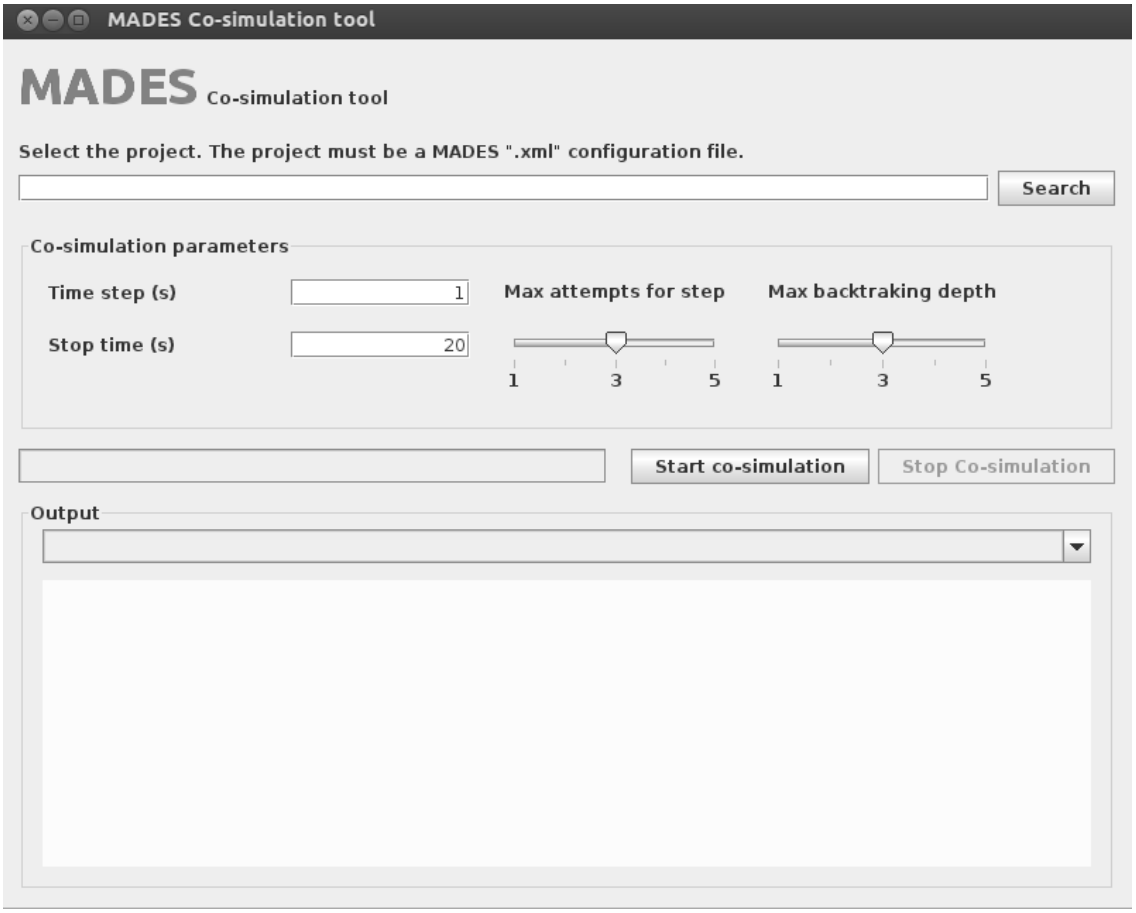

Figura 2.2 – Interface do co-simulador MADES modificado pela abordagem MCA.

Porém muitas vezes, um esforço significativo em termos de análise e transformações é necessário para obter a modelagem desta forma. Ela exige uma série de habilidades de mão de obra e é propensa a erros (ZUPANCIC et al., 2008).

A fim de permitir a reutilização de modelos, as equações devem ser fixadas de uma forma neutra, que é a chamada modelagem não causal (CELLIER, 1991). Causalidade é feita artificialmente, por exemplo, num resistor é difícil dizer se a corrente causa a tensão ou tensão causa a corrente (ZUPANCIC et al.,2008). Tal representação é feita na forma de EDA's (equações diferenciais algébricas):

$$F\left(y^{(n)}, y^{(n-1)}, y^{(n-2)}, \ldots, y', y, x\right) = 0$$

O equacionamento "natural" do sistema gera sempre equações que são então manipuladas para ter uma EDO (MATTSSON; ELMQVIST; OTTER, 1998). Ferretti, Magnani e Rocco (2004) citam: "É muito mais fácil, mais cômodo e mais natural, usar ferramentas de modelagem não causais como Dymola com Modelica". Então a álgebra computacional se

ocupa de conseguir um código de simulação eficiente semelhante às equações convertidas para a forma EDO manualmente, porém esta é desvantagem desta abordagem, apesar de facilitar a modelagem, ela piora o desempenho na simulação, algo que em sistemas muito complexos matematicamente pode levar a grandes atrasos de compilação.

A linguagem Modelica (MODELICA ASSOCIATION, 2003) foi desenvolvida exatamente para executar uma modelagem não causal e com isto é também uma linguagem orientada a objetos, portanto podem-se definir classes de modelos.

Os componentes que são importantes ao modelo podem ser definidos por meio da definição do sistema de equações de cada componente separadamente, sem a necessidade de "isolar" a variável a ser definida no programa (MATTSSON; ELMQVIST; OTTER, 1998). Isso permite definir e usar componentes separadamente e depois colocá-los juntos no modelo, e isso representa uma grande comodidade.

Porém a diferença mais importante no que diz respeito às ferramentas causais está na forma diferente de conexão destes componentes, por meio do uso de conectores (ZUPANCIC et al., 2008).

A conexão entre submodelos é baseada na conservação energia e de potencia das variáveis, que definem as relações adequadas e influência entre os movimentos, ou seja, ângulos, correntes, pressões, etc. Existem dois tipos de variáveis, as variáveis de esforço que devem ser iguais nos pontos de ligação (como temperatura, pressão, tensão, etc.) e as variáveis de fluxo cuja soma nos pontos de ligação é igual a zero (como corrente, força, etc.).

Para exemplificar, supõe-se modelar o circuito simples da Fig. 2.3, com dois componentes, um resistor e um gerador de corrente, e ao realizar os submodelos separadamente, obtêm-se como resultado a modelagem apresentada na Fig. 2.4.

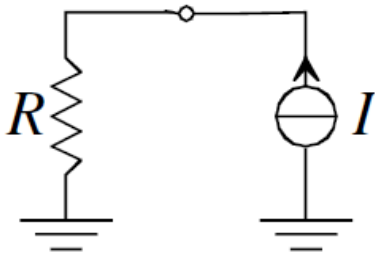

Figura 2.3 – Exemplo de um circuito elétrico simples com um resistor e um gerador de corrente.

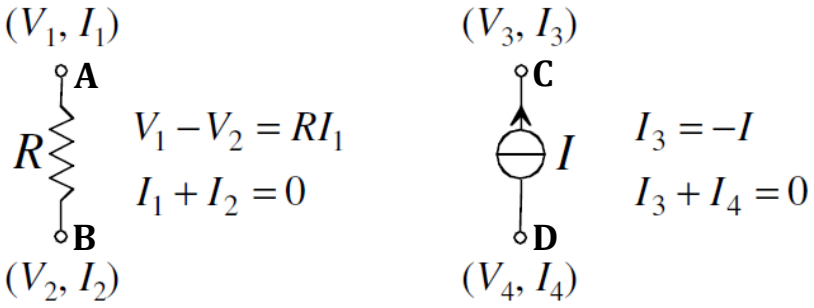

Figura 2.4 – Modelagem do resistor e do gerador de corrente separadamente.

Ao conectar o ponto A ao C, sabe-se que as tensões serão iguais, $V_1 = V_3$, ou seja, as variáveis de esforço são iguais, e a soma das correntes no nó deve ser nula, $I_1 + I_3 = 0$, ou seja, as variáveis de fluxo somam zero.

A diferença do Modelica, e da maioria das linguagens causais (ZUPANCIC et al., 2008), é que essa operação é feita automaticamente por meio dos "conectores", que são componentes previamente definidos da linguagem que identificam as variáveis que geram essas equações necessárias para ligar esses modelos. Com a sua utilização é possível simplificar os modelos, obtendo-os de maneira mais simples do ponto de vista de compreensão, assemelhando-se aos esquemas de modelagem feitos à mão (FRITZSON, 2004).

Como cada componente (por exemplo, um resistor), especificado por meio de um sistema de equações diferenciais algébricas, pode ser definido separadamente, é possível usar dos modelos já prontos e que já estão contidos na biblioteca incorporada ao programa. Usando deste fato, foram desenvolvidas diversas plataformas, entre elas o OpenModelica, que permitem que estes modelos sejam feitos de maneira gráfica, analogamente ao Simulink (FRITZSON et al., 2013).

Na Fig. 2.5 tem-se a comparação entre a modelagem no OpenModelica e no Simulink, com circuito elétrico como exemplo. Nota-se que o modelo no OpenModelica mantém a estrutura física do circuito, sendo assim mais fácil a compreensão do modelo. Em contrapartida o modelo no Simulink é baseado no fluxo de sinal, assim é mais difícil de compreender (FRITZSON et al., 2013).

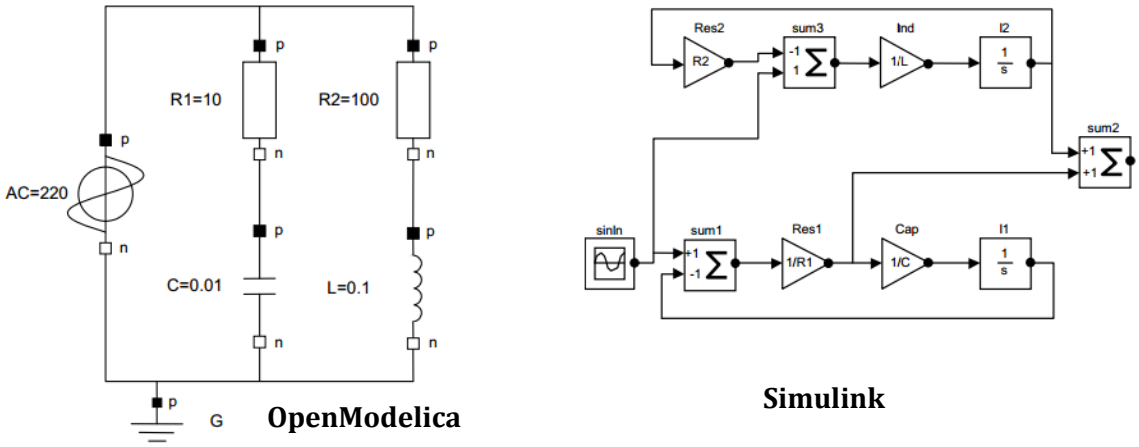

Figura 2.5 – Comparação entre a modelagem do mesmo sistema em OpenModelica e Simulink, extraída de (FRITZSON et al., 2013).

2.4. Modelagem utilizando Modelica/OpenModelica

Como a linguagem Modelica é não causal, a modelagem torna-se mais simples, pois é possível definir diretamente a equação do sistema. Para exemplificar será realizada a modelagem do circuito RC da Fig. 2.6.

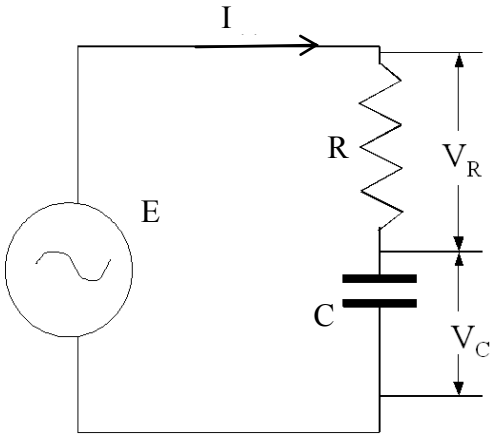

Figura 2.6 – Circuito RC utilizado como exemplo de modelagem

A partir da Fig. 2.6 têm-se as equações abaixo que, em Modelica, podem ser escritas como na Fig. 2.7:

$$\begin{cases} E = V_R + V_C \\ V_R = R.I \\ \dfrac{dV_C}{dt} = C.I \end{cases}$$

Figura 2.7 – Equações do circuito RC utilizado como exemplo de modelagem

Na Fig. 2.8 nota-se que a linguagem tem um espaço especifico para se escrever as equações, e que estas podem ser escritas de modo muito similar de como são obtidas. No exemplo, as constantes R (resistência) e C (capacitância) foram definidas de maneira diferentes para explicar o diferente uso das definições de *"constant"* e *"parameter"*. O uso de *"parameter"* permite alterações do valor da constante dentro do algoritmo, enquanto *"constant"* não permite. Outros detalhes da linguagem podem ser consultados na literatura de referência (FRITZSON, 2004).

```
model RC_simples
  constant Real R = 1;
  parameter Real C = 1;
  Real E = 1;
  Real Vr;
  Real Vc;
  Real I(start = 0);
equation
  E = Vr + Vc;
  Vr = R * I;
  der(Vc) = C * I;
end RC_simples;
```

Figura 2.8 – Modelo do circuito RC escrito em Modelica

Outra maneira de modelagem, usando-se do fato que esta linguagem ser orientada a objetos, é a modelagem separada dos componentes da Fig. 2.6 (Gerador, Resistor e Capacitor) e com o uso de conectores gerar o modelo completo do exemplo. Somado a este fato, vários componentes, como estes do exemplo, já estão presentes na biblioteca incorporada à linguagem, e o uso da plataforma OpenModelica permite realizar esta modelagem de maneira gráfica, tornando o este processo mais intuitivo. O resultado da modelagem no OpenModelica pode ser visto nas Figs. 2.9 e 2.10.

Nota-se que o modelo gráfico se assemelha muito ao sistema real e deve ser utilizado como complementação do método textual onde podem ser realizadas outras rotinas e algoritmos. Ao final o modelo é salvo em um arquivo ".mo" e se forem utilizadas classes definidas pelo usuário, recomenda-se que estas sejam escritas num mesmo arquivo para evitar a perda de informações.

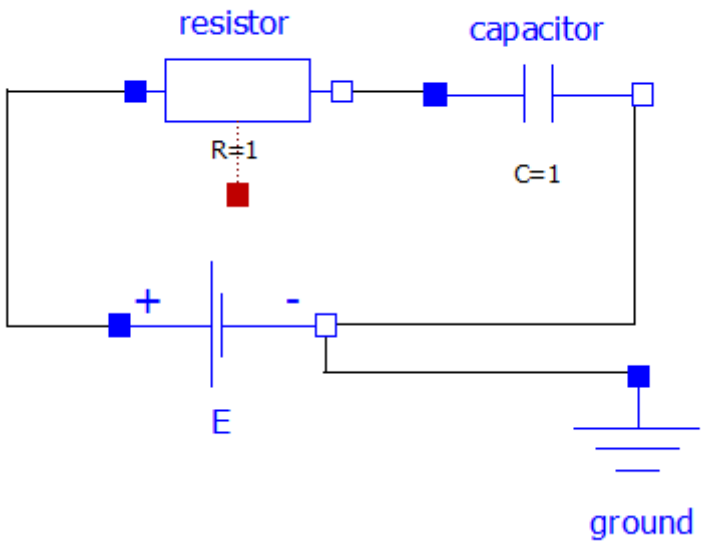

Figura 2.9 – Vista gráfica do modelo do circuito RC na plataforma OpenModelica

```
model RC_simples
  Modelica.Electrical.Analog.Basic.Resistor resistor(R = 1);
  Modelica.Electrical.Analog.Basic.Capacitor capacitor(C = 1);
  Modelica.Electrical.Analog.Sources.ConstantVoltage E(V = 1);
  Modelica.Electrical.Analog.Basic.Ground ground;
equation
  connect(ground.p,E.n);
  connect(capacitor.n,E.n);
  connect(E.p,resistor.p);
  connect(resistor.n,capacitor.p);
end RC_simples;
```

Figura 2.10 – Código gerado pelo uso da vista gráfica na modelagem do circuito RC na plataforma OpenModelica

2.5. Lógicas temporais e o simulador Zot

Na lógica clássica, as fórmulas são sempre avaliadas dentro de um único mundo fixo. Por exemplo, a proposição “A” do tipo “é domingo”, deve ser ou verdadeira ou falsa, e este valor deve ser fixo neste “mundo”.

Porém se esta proposição pode ser satisfeita ou não dependendo de determinadas condições (como é o caso), tem-se a expansão da lógica clássica para as chamadas lógicas modais, que possibilitam trabalhar dentro de um conjunto de mundos (ou realidades), onde em alguns “A” é verdadeira e em outros “A” é falsa (CHELLAS, 1980).

A lógica temporal é uma lógica modal que efetua esta avaliação (verdadeiro ou falso) em instantes de tempo e são estes instantes que constituem este conjunto de “mundos” possí-

veis, pois a cada instante uma proposição deve ser verdadeira ou falsa, mas nunca as duas coisas ao mesmo tempo (HASLE; OHRSTROM, 1995).

Considerando a declaração: "A máquina está parada". Embora seu significado seja constante em relação ao tempo, o valor-verdade da declaração pode variar com o tempo. Algumas vezes a declaração é verdadeira, e algumas vezes falsa, mas nunca será verdadeiro e falso ao mesmo tempo. Em lógica temporal, declarações podem ter um valor-verdade que variam com o tempo.

Resumindo, as lógicas temporais estendem a lógica clássica permitindo que as proposições possam ter um valor-verdade que variam com o tempo e também introduzindo novos operadores, com o objetivo de poder expressar eventos no tempo (passado, presente e futuro) (LACERDA E SILVA, 2011).

Na lógica clássica ou proposicional, têm-se os seguintes operadores básicos: Negação (¬), Ou (∨), E(∧), Se (→) e Se-e-somente-se (↔). Junto a estes, uma lógica temporal possui também operadores que auxiliam a definir quantidades no tempo (BERNINI, 2009):

- Next(A): é verdadeira se e somente se a "A" for verdadeira no instante sucessivo;
- Finally(p): é verdadeira se e somente se a "A" for verdadeira em algum instante no futuro;
- Always(p): é verdadeira se e somente se a "A" for sempre verdadeira no futuro;
- Until(p,q): esta condição é verdadeira se p for verdadeira enquanto q for verdadeira.

Baseados nestes conceitos têm-se uma série de lógicas derivadas, entre elas o LTL (Lógica temporal linear), que trabalha com o tempo de maneira linear e é a base da lógica chamada TRIO, que como definem Furia et al. (2012) ela consegue trabalhar de maneira quantizada com os instantes de tempo, ou seja, consegue definir por exemplo restrições em determinados instantes, o que não é possível na LTL.

O LTL é uma linguagem de primeira ordem que tem a capacidade de expressar as propriedades de uma forma concisa e intuitiva. Logo, o TRIO herda estas características e possui operadores temporais quantizados que se referem ao passado e ao futuro, o que permite definir propriedades cujo valor muda ao longo do tempo, não apenas suportar suas mudanças espontâneas como é o caso do LTL (FURIA et al., 2012).

Cada fórmula no TRIO assume um significado em relação ao instante de tempo, que é deixado implícito. O TRIO tem a mais os seguintes operadores: Futr(p,t) e Past(p,t), que su-

pondo que o instante atual seja T (implícito) e que são definidos segundo Bernini (2009) da seguinte maneira:

- Futr(p,t) : p é verdadeira no instante T+t;
- Past(p,t) : p é verdadeira no instante T-t.

A partir destes, uma série de outros operadores temporais foi derivada para simplificar o uso da linguagem, e os mais relevantes a este trabalho serão apresentados e explicados nos próximos capítulos.

Por fim, a plataforma onde são escritas estas regras é Zot, um programa que verifica a coerência das regras definidas em TRIO, ou seja, executa um algoritmo interno para verificar se todas as regras impostas têm uma solução possível, e também de acordo com suas definições logicas próprias, define como resultado uma dessas possíveis soluções (PRADELLA; MORZENTI; SAN PIETRO, 2007).

2.6. Modelagem por regras lógicas utilizando TRIO/Zot

O sistema (*software* embarcado) interage com o ambiente (sistema físico) por meio das informações por este fornecidas em variáveis compartilhadas, por exemplo por meio da leitura das medições de sensores de erro, que por meio da análise destas medições pode, por exemplo, definir um novo ganho, uma nova velocidade em regime ou uma nova trajetória de acordo com sua lógica interna.

O modelo de sistema é expresso como um conjunto de restrições no tempo, assim os requisitos e objetivos do sistema embarcado devem primeiramente ser adaptados para expressões "naturais" lógicas. Por exemplo, se a restrição é que a posição não tenha sobressinal, então se traduz isso na forma:

"Posição" ≤ "Posição de regime".

Porém falta um detalhe neste requisito, o tempo. É necessário sempre definir quando que este requisito (que depois será traduzido em uma regra) deve ser respeitado. Ela pode ser válida para todo instante, logo:

"Posição" ≤ "Posição de regime", a todo instante; ou

Sempre, "Posição" ≤ "Posição de regime".

Ou ser válida num instante específico:

"Posição" ≤ "Posição de regime", em t=3.

Ou ainda depender de outra condição:

Quando “Mover mecanismo” for detectado

então “Posição” ≤ “Posição de regime”.

Outro fator importante é que se deve evitar criar regras determinísticas, como do tipo:

Se “velocidade” é baixa então “ganho” =10.

É recomendável criar regras não determinísticas, deixando alguns pontos “livres”:

Se “velocidade” é baixa então 9 ≤ “ganho” ≤ 10.

Isto permite ao programa selecionar a melhor solução de acordo com a sua lógica interna e garante maior robustez ao modelo evitando que pequenas alterações nas condições do modelo do ambiente alterem significativamente o controle lógico (BERGERO; KOFMAN, 2010) e diminuindo a possibilidades que outra regra venha a criar uma situação de conflito que impossibilite o sistema a chegar a um resultado (FURIA; ROSSI, 2010).

Uma vez definidos todos os requisitos então a tradução para TRIO torna-se simples:

- Sempre, “A”; “A”, a todo instante: Alw(A);
- Se “A” então “B”: A → B;
- “A”, em t=3: Futr(A,3);
- “A” ocorreu nos últimos 3 instantes: Lasted(A,3);
- “A” ocorrerá em até 3 instantes: W

Outras regras de tradução podem ser consultadas na referência (BERNINI, 2009).

3. METODOLOGIA APLICADA

Como justificado na revisão bibliográfica, a metodologia utilizada foi a MCA, criada por Baresi et al. (2012) e baseada no programa MADES (BAGNATO et al., 2010). Esta metodologia define os seguintes passos e procedimentos que devem ser aplicados para a modelagem do sistema híbrido através da utilização da co-simulação:

1. Separação do sistema híbrido em ambiente e sistema;
2. Definição das variáveis compartilhadas, com a classificação para cada submodelo em variáveis de entrada e variáveis de saída;
3. Modelagem do ambiente por meio da plataforma OpenModelica;
4. Modelagem do sistema por meio de regras e restrições lógicas, escritas em TRIO e compiladas dentro da plataforma Zot;
5. Criação de regras lógicas para guiar a simulação de modo a não obter resultados "triviais";

Os arquivos dos modelos precisam ter todos o mesmo nome, com apenas a extensão diferente. Os resultados das co-simulações são exibidos na interface do programa e salvas num arquivo chamado MADESOutput.xml.

4. CASO DE ESTUDO

Para o caso de estudo foi selecionada a modelagem de um sistema de braços robóticos trabalhando simultaneamente no mesmo espaço de trabalho, de modo a continuar o trabalho desenvolvido por Bueno (2013), com a principal restrição de evitar colisões, um problema presente, por exemplo, num processo industrial de solda a ponto.

4.1. Modelo do ambiente

O modelo do ambiente é representado por um conjunto de braços robóticos que estão localizados no mesmo espaço de trabalho. Na modelagem de cada braço é considerado tanto o modelo físico do braço livre no espaço quanto o modelo do motor que impulsiona as articulações flexíveis.

Para cada braço define-se um modelo físico (Fig. 4.1) composto por quatro classes de objetos, "Braço", "Motores", "Controladores" e "Cinemática Inversa".

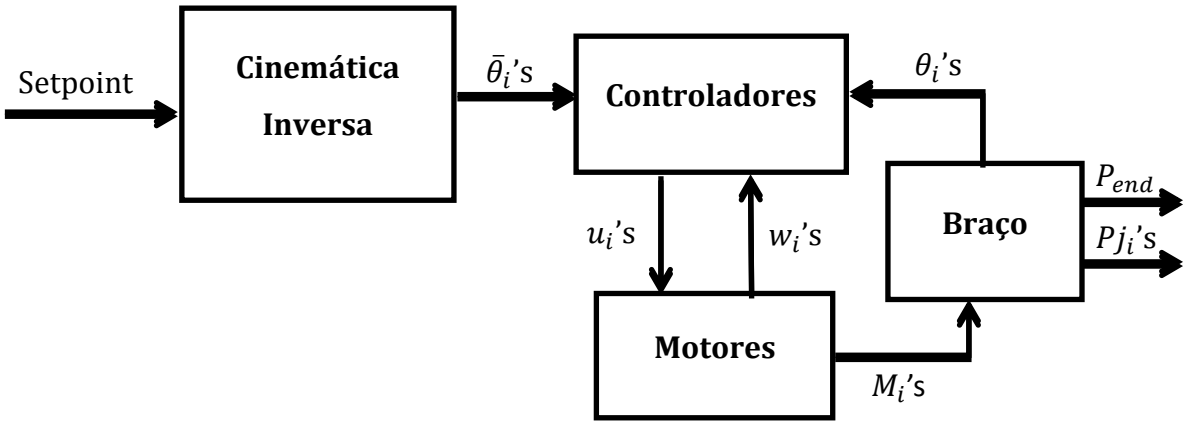

Figura 4.1 – Esquema de modelagem de um braço (BUENO, 2013).

A classe "Braço" define o braço livre no espaço, ou seja, nenhum torque aplicado nas articulações, expressando as relações cinemáticas e dinâmicas do braço. Esta classe tem como entrada o torque aplicado às articulações (M_i), e como saída os ângulos (θ_i), as posições espaciais das articulações (Pj_i) e a posição do efetuador (P_{end}).

A classe "Motores" define os motores que geram o torque para mover os segmentos (*links*). Podem ser motores DC *brushless* ou motores de passo, isto é indiferente ao projeto. A outra classe ("Controladores") representa o controlador dos motores que usa o erro angular e gera as tensões que devem ser aplicadas aos respectivos "Motores".

Por último tem-se um sistema que lida com a cinemática inversa (a classe “Cinemática Inversa”). Esta parte é necessária para obter as posições angulares de cada junta a partir da posição final, porque o *Setpoint* dado é a posição do efetuador (embora se possa controlar apenas os ângulos das juntas).

Existem duas maneiras de realizar a cinemática inversa: cálculo numérico e o cálculo analítico. Para sistemas complexos, é mais adequada a primeira solução, por meio do cálculo da matriz Jacobiana (SCIAVICCO; SICILIANO, 2000). No presente estudo, porém, utiliza-se o cálculo analítico porque o sistema é simples.

Para o estudo foi escolhido um braço com dois graus de liberdade (g.d.l.) capaz de realizar um movimento planar (ou seja, sem efeitos gravitacionais), para assim obter analiticamente a cinemática inversa do braço (Fig. 4.2).

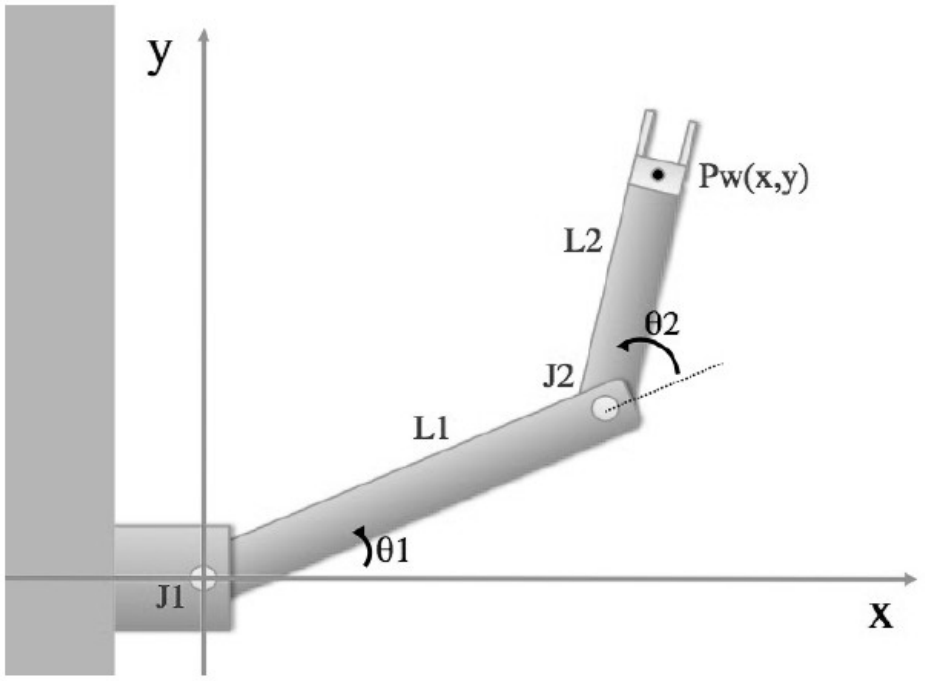

Figura 4.2 - Diagrama de um braço articulado com 2 g.d.l. e movimento planar, extraída de (BUENO, 2013).

O braço é constituído por duas articulações ou juntas (J1 e J2) onde atuam binários adicionais dos motores para mover respectivamente os segmentos L1 e L2.

Os motores escolhidos foram os DC *brushless*, esquematicamente definidos como um gerador de torque, uma constante de tempo em cascata e uma caixa de redução, como pode ser visto a partir do esquema representado em OpenModelica mostrado na Fig. 4.3.

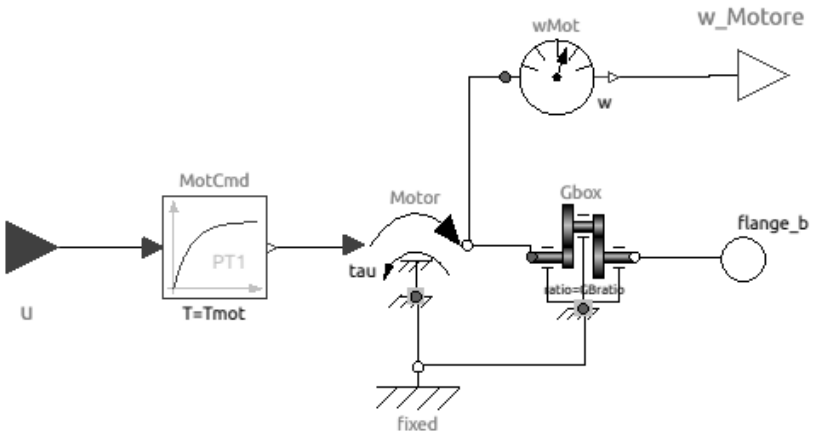

Figura 4.3 - Diagrama em OpenModellica de cada motor (BUENO, 2013).

O trabalho de Ferretti et al. (2003) mostra que dada a simplicidade, pode-se projetar o controlador como dois controladores simples desacoplados relativos à posição angular de cada motor e cada controlador como um PI (proporcional-integral) sobre a velocidade do motor em cascata com um P (Proporcional) sobre a posição, representado na Fig. 4.4.

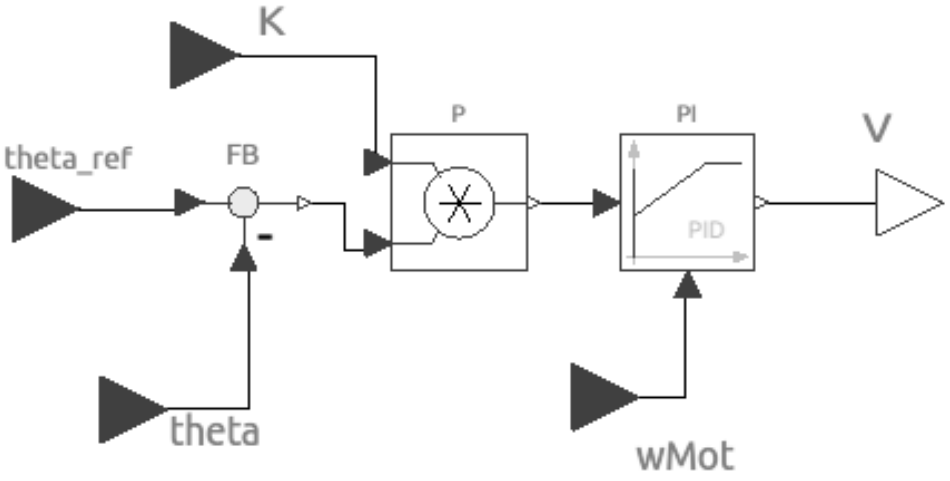

Figura 4.4 – Diagrama em OpenModellica do controlador atuante em cada motor (BUENO, 2013).

Projetado deste modo, o sistema conjunto “Controladores”+“Motores” funciona de modo a impor ao sistema o ângulo desejado em cada junta.

Deseja-se que o desempenho do controlador seja tal a para garantir uma resposta rápida sem qualquer sobressinal.

De modo a obter que a resposta angular seja a mais rápida possível sem sobressinal, foram definidos como parâmetros do regulador PI, os valores $K_p = 20$ e $T_i = 0.05$ s, de modo que, inicialmente, o ganho K do regulador P, mantém-se constante e igual a 1.

Para o cálculo dos ângulos desejados, utilizando a cinemática inversa analítica, que neste caso é reduzida com as seguintes equações (extraídas do livro de Sciavicco e Siciliano, (2000)):

$$B = \sqrt{(SP_x - O_x)^2 + \left(SP_y - O_y\right)^2}$$

$$q_1 = \operatorname{atan2}\left(SP_y - O_y, SP_x - O_x\right)$$

$$q_1 = \operatorname{acos}\left(\frac{L_1^2 - L_2^2 + B^2}{2.L_1.B}\right)$$

$$\bar{\theta}_1 = q_1 + q_2$$

$$\bar{\theta}_2 = -\pi + \operatorname{acos}\left(\frac{L_1^2 + L_2^2 - B^2}{2.L_1.L_2}\right)$$

onde:

- (SP_x, SP_y) é o *setpoint* desejado;
- (O_x, O_y) é a origem do braço (posição da primeira articulação, importante no caso em que se consideram mais braços juntos);
- L_1 e L_2 são os comprimentos dos segmentos 1 e 2.

Ao final tem-se o modelo do próprio braço, sem ainda a atuação de qualquer torque sobre as juntas, que pode ser visto na Fig. 4.5.

Neste modelo, há um componente de translação (origem) para conseguir usar este mesmo modelo quando se colocam mais braços a trabalharem juntos. As articulações (Joint_1 e Joint_2) são de rotação no eixo z, então não está presente a ação da força gravitacional. As articulações são também caracterizadas por um amortecimento e uma rigidez, escolhidas de acordo com os valores geralmente utilizados e obtidos do trabalho de Ferreti, et al. (2003).

Após reunir todas as classes listadas anteriormente, como mostrado na Fig. 4.1, chega-se ao modelo do braço controlado, Fig. 4.5. Na entrada deste sistema físico têm-se as seguintes variáveis:

- As posições finais desejadas do efetuador (Alvo), que se tratam do objetivo do movimento do braço;

- Os ganhos dos controladores ligados aos ângulos de junta, porque com isso pode-se gerir melhor o movimento do braço, diminuindo a ação de controle ou bloqueando o movimento do braço, dada qualquer perturbação indesejada.

As saídas são as posições das articulações e do efetuador que são importantes para os cálculos dos erros e de possíveis colisões.

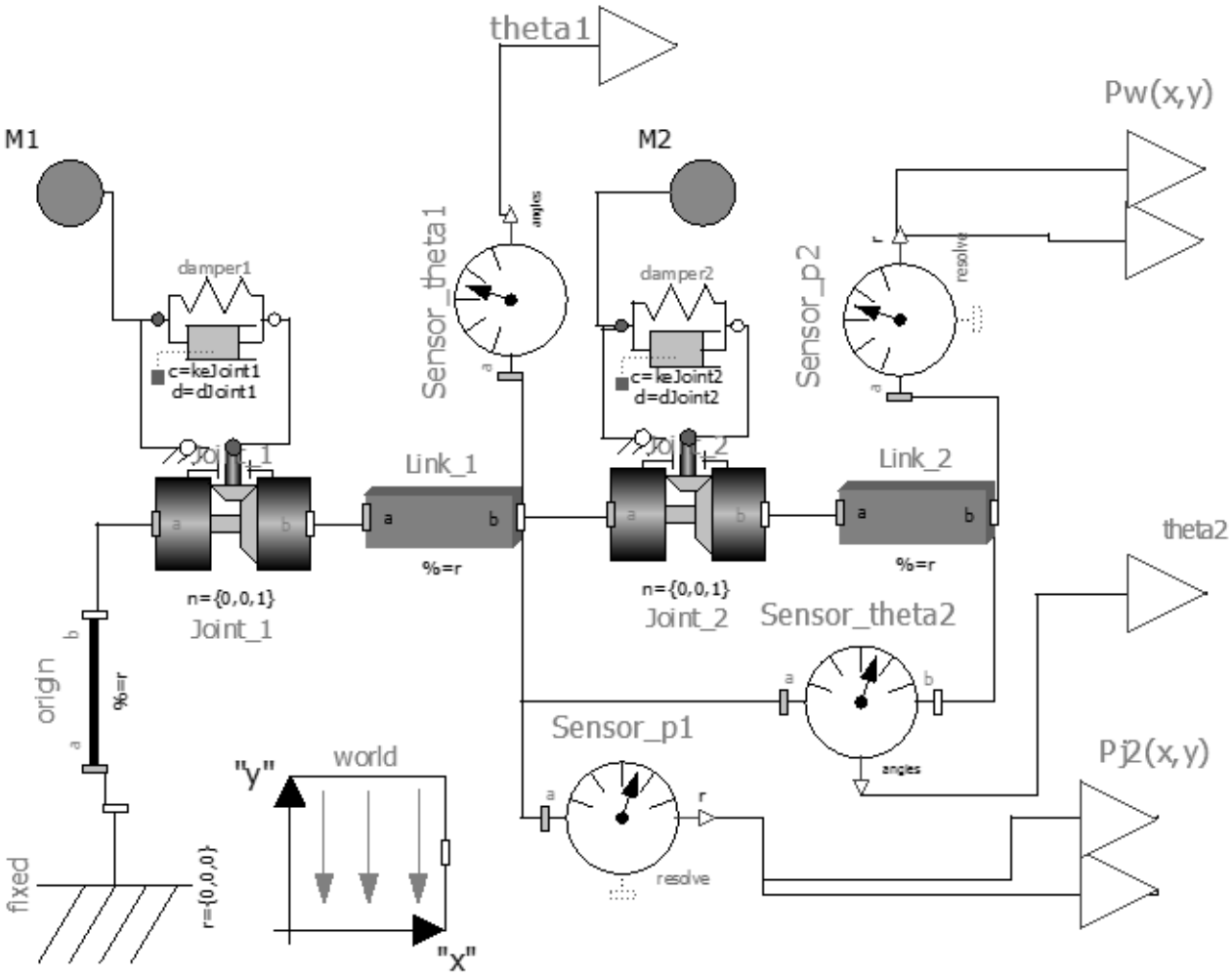

Figura 4.5 – Esquema em OpenModellica de um braço (BUENO, 2013).

Com este modelo do braço controlado anteriormente descrito, pode-se inserir dois ou mais braços numa mesma área de trabalho, por meio do posicionamento das respectivas origens em pontos diferentes, num esquema que pode ser representado pela Fig. 4.6.

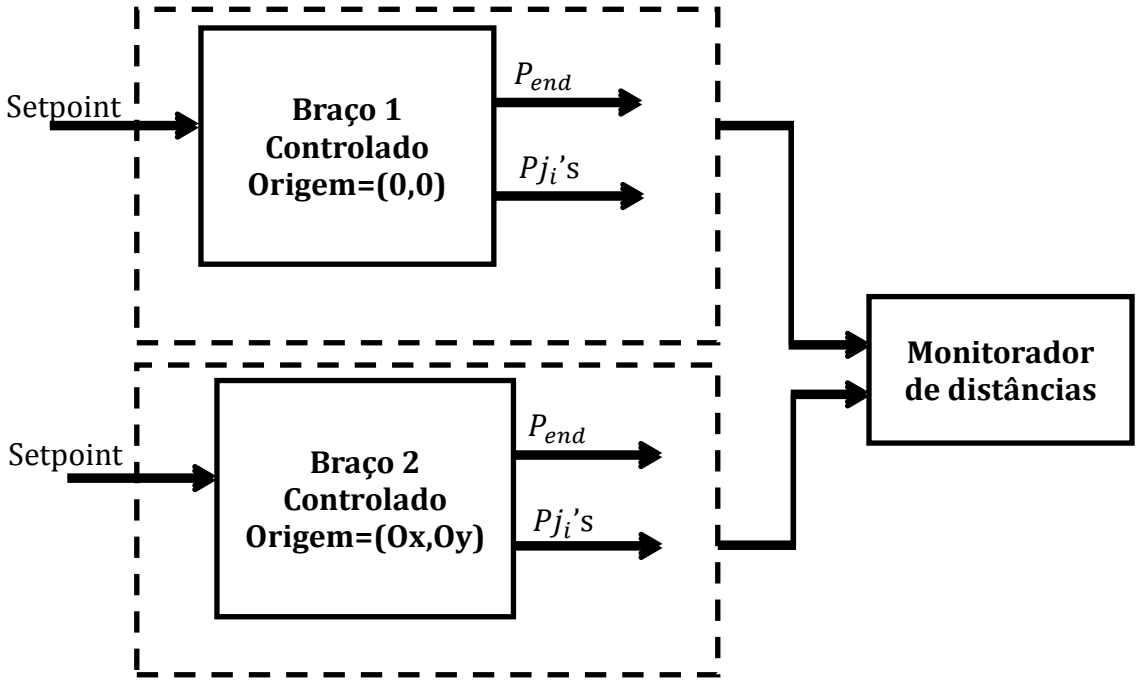

Figura 4.6 - Diagrama do sistema com dois braços (BUENO, 2013).

Nota-se a partir da Figura 4.6 que, quando se decide colocar dois ou mais braços juntos, é importante ter em conta a distância entre eles para poder efetuar qualquer controle de segurança que vise evitar colisões.

E é exatamente para a politica para o calculo e estimativa destas distâncias que se encontram os desafios do projeto. Este desafio foi tratado por Bueno (2013), que utilizou vários métodos, e como melhor solução tem-se que se cada braço dispõe da posição da origem, junta e efetuador do outro, pode-se utilizar um método, baseado na distância de ponto a segmento de reta, com a correção de quando o ponto não está sobre o segmento.

O algoritmo pode ser descrito da seguinte maneira, utilizando para exemplo o caso do ponto do ponto Pw do braço 1 (Pw1 nas equações) em relação a L2 do braço 2, que é limitado pelos pontos Pw e Pj do braço 3, (Pw2 e Pj2 nas equações):

1. Calcula-se a distância deste ponto aos segmentos do outro braço:

$$d_r = \frac{\text{abs}\,(a.\,Pw1_x + b.\,Pw1_y + c)}{\sqrt{a^2 + b^2}}$$

$$a = Pw2_y - Pj2_y \;;\; b = -Pw2_x + Pj2_x \;;\; c = Pw2_x.\,Pj2_y - Pw2_y.\,Pj2_x$$

2. Então se calcula a distância deste ponto (Pw1) aos pontos limitantes do segmento (Pj2 e Pw2):

$$d_1 = \sqrt{(Pj2_x - Pw1_x)^2 + \left(Pj2_y - Pw1_y\right)^2}$$

$$d_2 = \sqrt{(Pw2_x - Pw1_x)^2 + \left(Pw2_y - Pw1_y\right)^2}$$

3. Seja M, a projeção do ponto sobre a reta definida pelo segmento L2, então se calcula a distância de M aos limitantes (Pj2 e Pw2), que pode ser resumida como:

$$d_{m1} = \sqrt{\mathrm{d}_1^2 - d_r^2} \text{ e } d_{m2} = \sqrt{\mathrm{d}_2^2 - d_r^2}$$

4. Verifica-se se o ponto M está no segmento L2, ou fora, e determina-se a distância efetiva (d) entre Pw2 e L2, pelo algoritmo da Fig. 4.7;
5. Repete-se os passos 1 a 4 para origem, juntas e efetuador de todos os braços e em relação a cada segmento, que no caso deste estudo serão 12 distâncias calculadas;
6. Obtém-se o mínimo destas distâncias efetivas que será a distância considerada entre os braços.

```
//Verifica se está mais próximo de Pj2
If dm2 > dm1
        // Verifica se M está fora de L2
        If dm2 > L2 then d = d1
        Else d = dR
Else
        // Verifica se M está fora de L2
        If dm1 > L2 then d = d2
        Else d = dR
```

Figura 4.7 – Algoritmo para cálculo da distância efetiva, baseado em (BUENO, 2013).

4.2. Modelo do sistema

O modelo do sistema envolve os requisitos e objetivos que se deseja ao sistema híbrido, ou seja, as regras que serão escritas envolvem o planejamento dos cenários que se deseja testar, assim os cenários que serão testados são:

1. Utilização de um controle que torne o sistema mais rápido;
2. Adição de um vínculo de segurança que exprime a necessidade de alterar o movimento do robô se no seu raio de ação está presente uma pessoa ou qualquer outro obstáculo genérico;
3. Adição uma restrição que expressa a necessidade do efetuador ter tempo de fechar-se sobre um objeto no alvo;
4. Ao reunir mais braços, adicionar uma restrição com objetivo é evitar a colisão entre eles.

Com os cenários definidos, pode-se iniciar a projetar as regras que atendem a estes cenários.

4.2.1. 1° Cenário

Analisando o objetivo do 1° cenário, conclui-se que se deve encontrar uma estratégia para atuar sobre os ganhos dos controladores proporcionais de posição de maneira a alcançar este objetivo.

Sabe-se que ao aumentar o ganho, diminui-se o tempo de resposta do sistema, mas também se aumenta o sobressinal (OGATA, 2011). Utilizando-se disto pode-se concluir que se o sistema ainda está longe do seu valor a regime, um ganho grande traria apenas vantagens, pois o sobressinal é uma característica avaliada quando o sistema ultrapassa o valor de regime. Assim, como neste sistema o valor a regime será o alvo, tem-se a conclusão expressa pela Tab. 4.1:

Tabela 4.1 – Relação qualitativa entre a distância do alvo e ganho

Distância	**Ganho**
Longe do alvo	Ganho grande
Perto do alvo	Ganho pequeno

Tem-se então que o ganho deve ser de certa forma proporcional ao erro (que expressa a distância ao alvo), estabelecendo um escalonamento discreto de ganhos, que é uma forma de controle “adaptativo” (ÅSTRÖM; WITTENMARK, 1995).

A técnica de escalonamento discreto de ganhos estabelece uma relação entre o erro e o ganho, que ao invés de ser linear, é uma política a ser adotada com o ganho para cada faixa de erro. Essa politica pode ser definir um valor para o ganho (Fig. 4.8) ou uma faixa de valores (Fig. 4.9). Nas figuras 4.8 e 4.9 tem-se um exemplo para o caso de dois intervalos de erro, onde nota-se que o caso da Fig. 4.8 é determinístico e o caso da Fig. 4.9 é não determinístico, pois o ganho pode assumir qualquer valor dentro dos intervalos (em cinza na Fig. 4.9).

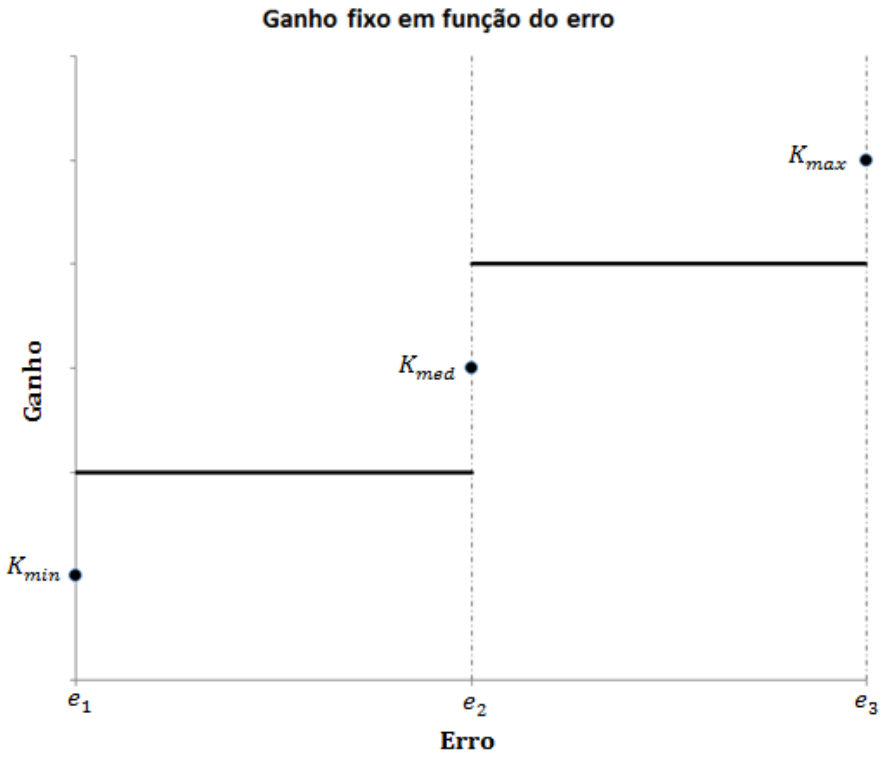

Figura 4.8 – Exemplo de política de ganho fixo em cada intervalo

A política escolhida foi aquela demonstrada na Fig. 4.9, exatamente por dar liberdade de escolha ao sistema de controle. Assim definidos os intervalos de erro e ganho ter-se-iam regras da forma:

$$e_i > err > e_j \rightarrow K_i < g < K_j \qquad (1)$$

Onde:

- err é o erro medido no sistema;
- g é o ganho do controlador;

- e_i e e_j são os limites de um cada intervalo de erro;
- K_i e K_j são os limites do respectivo intervalo de ganhos definido para esse intervalo de erros.

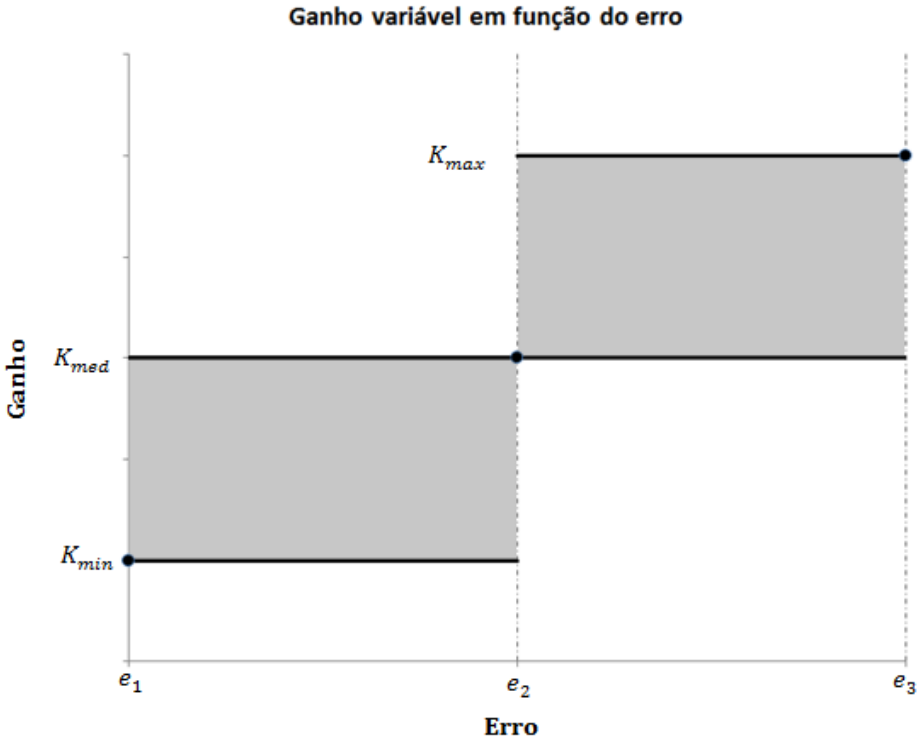

Figura 4.9 - Exemplo de política de ganho variável em cada intervalo

Para fazer estas alterações, deve-se avaliar se o sistema não efetua alterações desnecessárias no ganho que ocorreriam quando o sistema já está "rápido". Esse fenômeno é inversamente proporcional ao intervalo de amostragem adotado. Assim adota-se verificar o intervalo do erro nos últimos T_{err} instantes onde T_{err} é adotado inicialmente como 5 u.t. De acordo com o que já foi relatado, define-se a seguinte regra:

$$\text{Lasted}\left(e_i > err > e_j\,, T_{err} \rightarrow K_i < g < K_j\right) \qquad (2)$$

Estas regras se baseiam na operação de um único controlador, mas são válidas e aplicadas a ambos os controladores ao escrever o código.

Para este caso de uso serão utilizados os intervalos fornecidos na Tab. 4.2, com a gama de valores retirados do estudo de Rocco et al., (2011) e com a adequação a este caso de uso por meio de pequenas simulações com o braço, sabendo os limites de saturação do regulador:

Tabela 4.2 – Definição dos intervalos de erro e respectivos intervalos de ganho

Erro	Ganho
err > 90°	6< g < 10
60° < err < 90°	4< g < 6
10° < err < 60°	2< g < 4
err < 10°	1 < g < 2

4.2.2. 2° Cenário

No 2° cenário tem-se um vínculo que exprime a necessidade de alterar o movimento do robô se no seu raio de ação está presente uma pessoa ou qualquer outro obstáculo genérico. Este, portanto, não pode continuar seguindo normalmente a sua a trajetória pré-estipulada. É importante notar que o objetivo de segurança muda se no espaço de trabalho do robô está presente uma pessoa ou um obstáculo, na presença de uma pessoa no espaço de trabalho é mais importante à proteção de segurança da pessoa, enquanto que na presença de um obstáculo, a segurança do robô é mais importante (que pode ser danificado pelo impacto).

Neste ponto e para esta situação, é necessário fazer uma escolha. De fato o robô pode retardar o seu movimento para não colidir com o corpo ou pode parar até que a posição do obstáculo não seja mais próxima a este. Ficar certo tempo nesta condição parece ser a melhor opção, mesmo que o sensor já não sinta a proximidade do corpo/obstáculo.

Escolheu-se a segunda opção, e se propõe a seguinte regra, onde *DontMove* é um predicado que força o braço não se mover por $\mathrm{T_{person}}$ instantes que para as simulações será de 5 u.t.:

$$\mathrm{PersonDetected} \rightarrow \mathrm{Lasts}\left(\mathrm{DontMove}\,,\mathrm{T_{person}}\right) \qquad (3)$$

4.2.3. 3° Cenário

Da mesma forma o 3° cenário pode ser definido como uma restrição que o robô depois que chegar ao alvo, deve manter-se parado nesta posição durante um intervalo de tempo definido, que para as simulações também será de 5 u.t.

Esta regra é expressa pelo conjunto de regras (4) a (7), onde OnTarget é um predicado que informa se o robô atingiu o alvo, OnTgx e OnTgy são predicados criados para saber se o efetuador é suficientemente próximo ao Alvo em “x” e em “y”, e “eps” é a tolerância do sistema:

$$\text{OnTarget} \rightarrow \text{Lasts}\left(\text{DontMove}\,, \text{T}_{\text{ontarget}}\right) \quad (4)$$

$$\text{OnTarget} \leftrightarrow \text{Lasted}(\text{OnTgx} \wedge \text{OnTgy}\,, 1) \quad (5)$$

$$\text{OnTgx} \leftrightarrow (-\text{eps} < (\text{Pw}_\text{x} - \text{SP}_\text{x}) < eps\,) \quad (6)$$

$$\text{OnTgy} \leftrightarrow \left(-\text{eps} < \left(\text{Pw}_\text{y} - \text{SP}_\text{y}\right) < eps\,\right) \quad (7)$$

Entretanto a regra DontMove pode ser definida de duas maneiras diferentes, com o objetivo de anular a tensão fornecida ao motor: anulando os ganhos ou alterando o valor do *setpoint*, ou ainda colocando a posição atual como o novo *setpoint*.

Como esta regra tem prioridade sobre todas as outras, por ser um vinculo de segurança, as regras não prioritárias devem possuir o seguinte vinculo, onde o predicado *Regra_nao_Prioritaria* engloba todas as regras que não estão relacionadas às regras de segurança:

$$\neg\text{DontMove} \rightarrow \text{Regra\_nao\_Prioritaria} \quad (8)$$

A avaliação da melhor política para regra DontMove será um cenário à parte e portanto convém definir as duas possibilidades do cenário, sendo a regra (9) para o primeiro e as regras (10) e (11) para o segundo:

$$\text{DontMove} \rightarrow (\text{g}_1 = 0 \wedge \text{g}_2 = 0) \quad (9)$$

$$\text{DontMove} \rightarrow \left(\text{SP}_\text{x} = \text{Pw}_\text{x} \wedge \text{SP}_\text{y} = \text{Pw}_\text{y}\right) \quad (10)$$

$$\neg\text{DontMove} \rightarrow \left(\text{SP}_\text{x} = \text{Tg}_\text{x} \wedge \text{SP}_\text{y} = \text{Tg}_\text{y}\,\right) \quad (11)$$

Onde:

- g_1 e g_2 são respectivamente os ganhos do 1° e do 2° controlador;
- SP_x e SP_y são os *setpoints* que serão mandados ao modelo do ambiente;
- Pw_x e Pw_y são as posições atuais do efetuador;
- Tg_x e Tg_y é o alvo desejado, definido pelo usuário.

Esta regra impõe a necessidade de um vínculo de integridade da simulação, caso contrario, o programa tem a possibilidade de escolher este predicado como verdadeiro sempre e nesta condição o robô não se move. Então deve ser definido que esta condição seja verdadei-

ra somente quando ocorreu uma das duas condições: o braço atingiu o alvo, ou uma pessoa/obstáculo foi detectada próxima ao braço robótico, e ter-se-á a regra (12):

$$\text{DontMove} \leftrightarrow (\text{OnTarget} \vee \text{PersonDetected}) \qquad (12)$$

4.2.4. 4° Cenário

No 4° cenário, com a reunião de mais braços ao sistema devem-se adicionar outros vínculos para controlar a iteração dos braços no mesmo espaço de trabalho. O objetivo é evitar a colisão dos braços, de modo que é necessário encontrar algum algoritmo para estima e cálculo da distância entre eles, analisando a possibilidade que muitas vezes o controlador terá apenas as posições do efetuador para tomar a decisão de haverá uma colisão ou não. Isto foi comentado na modelagem do ambiente e foi mais bem estudado em Bueno (2013).

Uma vez definido este método para estima das distâncias, pode-se definir uma política a se tomar quando uma colisão eminente for detectada pelo sistema.

A primeira opção é parar o movimento, ou seja, quando uma dada distância é menor que uma tolerância, o movimento dos braços será bloqueado, onde dist_{12} é a distancia entre os braços 1 e 2, DontMove_1 e DontMove_2 são respectivamente as restrições DontMove dos braços 1 e 2:

$$\text{dist}_{12} < dist_{\min} \rightarrow (\text{DontMove}_1 \wedge \text{DontMove}_2) \qquad (13)$$

Como política de segurança, para evitar danos aos braços, esta regra pode ser bem vista, mas ela impede que alguns movimentos possíveis sejam realizados. Por exemplo, se na condição inicial e final os braços não estão sobrepostos, então existe um caminho que leve da condição inicial à final sem colisão. Porém, com esta política, assim que for detectada uma colisão, o sistema para e não tenta outra rota possível.

Assim, como alternativa, propõe-se uma politica baseada no seguinte raciocínio: se a distância entre os braços for pequena, mas ainda não criticamente pequena, entra-se numa zona limite, caracterizada pela distância entre os braços estar entre dois valores limites. Nesta situação um dos braços continuaria o seu movimento e o outro iniciaria a retornar à posição inicial, até que saia desta zona limite e entre na zona crítica ou volte a situação normal. Isto é expresso da seguinte maneira:

1. O sistema inicia o movimento dos dois braços

$$\text{SitNormal} \rightarrow \text{Alvo}_1 \wedge \text{Alvo}_2 \wedge \neg\text{Inicio}_1 \wedge \neg\text{Inicio}_2 \quad (14)$$

$$\text{Alvo}_1 \rightarrow (\text{SP}_x)_1 = (\text{Tg}_x)_1 \wedge \left(\text{SP}_y\right)_1 = \left(\text{Tg}_y\right)_1 \quad (15)$$

$$\text{Alvo}_2 \rightarrow (\text{SP}_x)_2 = (\text{Tg}_x)_2 \wedge \left(\text{SP}_y\right)_2 = \left(\text{Tg}_y\right)_2 \quad (16)$$

$$\text{Inicio}_1 \rightarrow (\text{SP}_x)_1 = (\text{Pw0}_x)_1 \wedge \left(\text{SP}_y\right)_1 = \left(\text{Pw0}_y\right)_1 \quad (17)$$

$$\text{Inicio}_2 \rightarrow (\text{SP}_x)_2 = (\text{Pw0}_x)_2 \wedge \left(\text{SP}_y\right)_2 = \left(\text{Pw0}_y\right)_2 \quad (18)$$

Onde SitNormal é o predicado que define que o sistema está se movendo normalmente, Alvo 1 e Alvo 2 definem como *Setpoint* (variáveis SP's) aos braços os valores dos alvos definidos pelo usuário (variáveis Tg's) e Inicio1 e Inicio2 definem como *Setpoint* (variáveis SP's) aos braços os valores das condições iniciais (variáveis Pw0's);

2. Ao detectar a distância pequena entre os braços, entra na 1ª zona limite e o braço 2 recebe como *Setpoint* a condição inicial

$$\text{dist}_{12} < dist_{\text{lim}} \rightarrow \text{ZonaLim}_1 \quad (19)$$

$$\text{ZonaLim}_1 \rightarrow \text{Inicio}_2 \wedge \neg\text{Alvo}_2 \quad (20)$$

3. Se o sistema detecta que a distância ficou maior que a limite, então sai da zona limite e entra na condição normal.

$$Lasted(\text{ZonaLim}_1, 1) \wedge \text{dist}_{12} > dist_{\text{lim}} \rightarrow \neg\text{ZonaLim}_1 \wedge \text{SitNormal} \quad (20)$$

4. Se o sistema detecta que a distância ficou menor que a crítica, então sai da zona limite e na zona crítica, onde o movimento do braço 1 não pode mais continuar senão será inevitável a colisão, e assim o braço 1 retornará a posição inicial.

$$Lasted(\text{ZonaLim}_1, 1) \wedge \text{dist}_{12} < dist_{\text{crit}} \rightarrow \text{ZonaCrit}_1 \quad (21)$$

$$\text{ZonaCrit}_1 \rightarrow \text{Inicio}_1 \wedge \neg\text{Alvo}_1 \quad (22)$$

5. Assim que o braço 1 sair da zona limite, o braço 2 inicia seu movimento seguindo processo de 2 a 4.

6. Quando um dos braços chega ao objetivo, por exemplo o 1°, usa-se o predicado *On-Target_1* para informar ao 2° braço que pode iniciar seu movimento e neste caso o 1° braço começa a ser visto pelo 2° como um obstáculo apenas, parando seu movimento quando estiver muito perto deste, ou seja, o braço 1 gera o sinal de *PersonDetected* para o braço 2 quando a distância for menor que a crítica.

$$\mathrm{dist}_{12} < dist_{\mathrm{crit}} \rightarrow \mathrm{PersonDetected} \qquad (23)$$

4.3. Planejamento dos Testes Experimentais

4.3.1. 1° Cenário

Para os testes iniciais com um braço (1 e 2) foram analisadas duas situações com alvos diferentes, uma onde se muda apenas θ_1 enquanto θ_2 permanece constante, e outra onde ambos os ângulos mudam e os controladores devem trabalhar juntos.

Com a posição inicial do efetuador de Pw = (2, 0), os testes escolhidos estão na Tab. 4.3.

Tabela 4.3 – Testes escolhidos para avaliar o desempenho com as regras sobre o ganho

Teste	**Alvo (px,py)**	$\boldsymbol{\theta_1}$**(°)**	$\boldsymbol{\theta_2}$**(°)**
1	(0,2)	90	0
2	(0,1)	150	-120

Para expressar estas situações derivam regras lógicas que guiam o sistema, por exemplo para o 1° Teste tem-se que:

$$\mathrm{Alw}\left(\mathrm{SP_x} = 0 \wedge \mathrm{SP_y} = 2 \wedge \mathrm{g_1} = 1 \wedge \mathrm{g_2} = 1\right) \qquad (24)$$

Neste caso é necessário expressar que esta condição deve ser valida em todos os instantes de simulação (Operador “Alw”), pois esta é uma regra que estará nas restrições da simulação, fora do modelo do sistema, onde toda regra já é definida para todo instante. Para o segundo teste a regra é análoga.

Todos os testes serão realizados com um período de amostragem de 1s e durante um período de 30 u.t.

Para verificar que o modelo do ambiente foi executado corretamente, o bom funcionamento do co-simulador e para obter os dados que serão utilizados mais tarde para compara-

ção, foi necessária a execução de um teste “sem” controle, ou seja, apenas com o controle interno ao braço, sem adição de quaisquer alterações. Assim foram realizados 2 grupos de teste, no 1° grupo de testes, os testes “sem” controle e no 2° grupo de testes os testes com o controle adaptativo.

É importante salientar que é possível que o efetuador já esteja em uma posição estável, mas que, entretanto, as juntas apresentem ainda alguma oscilação, e logo não se possa considerar esta situação como estável (SCIAVICCO; SICILIANO, 2000). Assim para evitar isto, serão observados também os gráficos das posições das juntas.

Para o sistema em estudo, não é necessário observar as posições das juntas, porém enfatiza-se a importância deste raciocínio para sistemas mais complexos.

4.3.2. 2° Cenário

No 3° grupo de testes, referente ao 2° cenário, foi escolhido o alvo de (0, 1) e que o sensor detecta a presença de uma pessoa no instante 2 u.t. e a duração da co-simulação será reduzida para 10 u.t., dado que neste instante espera-se que o sistema, após a aplicação do controle adaptativo, já esteja em regime.

Foi realizado então um 4° grupo de testes para avaliar qual melhor política para a regra de DontMove.

É importante considerar que as regras anteriormente definidas não exigem o sinal DontMove aconteça apenas após a detecção por exemplo de uma "pessoa próxima". Para solucionar este problema, se necessário será imposta a seguinte restrição:

$$\mathrm{Alw}\left(\mathrm{DontMove} \rightarrow \mathrm{WithinPast}\left(\mathrm{PersonDetected}, \mathrm{T_{person}}\right)\right) \qquad (25)$$

Ela exige que o sinal DontMove só possa ocorrer se em algum dos últimos $\mathrm{T_{person}}$ instantes ocorreu PersonDetected.

4.3.3. 3° Cenário

No 5° grupo de testes, referente ao 3° cenário escolheu-se a seguinte regra sobre o alvo para "guiar" a simulação de maneira a ter uma mudança de alvo:

$$\mathrm{Lasts}\left(\left(\mathrm{SP_x} = 1 \wedge \mathrm{SP_y} = 0\right), \mathrm{T_{set}}\right) \wedge \mathrm{Futr}\left(\mathrm{Lasts}\left(\left(\mathrm{SP_x} = 2 \wedge \mathrm{SP_y} = 0\right), 20\right), \mathrm{T_{set}}\right) \qquad (26)$$

O tempo T_{set} foi escolhido de modo a ser capaz de ver que, com a regra do alvo, o sistema não reage instantaneamente, mas só depois de 10 u.t.

Foi também adicionada uma regra semelhante a (25) e não exatamente a mesma, apenar modificando o nome das variáveis envolvidas, para que as duas possam ser utilizadas simultaneamente, da seguinte forma:

$$\mathrm{Alw}\left(\mathrm{DontMove} \rightarrow \mathrm{WithinPast}\left(\mathrm{OnTarget}, \mathrm{T_{ontarget}}\right)\right) \qquad (27)$$

4.3.4. 4° Cenário

Para o 4° cenário, com os sistemas com múltiplos braços robóticos posicionados no mesmo espaço de trabalho, os problemas são ligados a encontrar um bom método para o cálculo, medição ou determinação da distância entre eles (BUENO, 2013) e, sucessivamente, definir claramente a política a tomar em relação aos mesmos, que foi definida no modelo do sistema.

Os testes são feitos para dois braços, mas podem ser expandidos para outros. O primeiro braço terá como origem local a origem do espaço de trabalho (0, 0) e o segundo braço estará na posição (1.5, 0). Ambos os braços começarão na posição vertical, ou seja, Pw1 (posição do efetuador do primeiro braço) é inicialmente (0, 2) e Pw2 (posição do efetuador do segundo braço) é (1.5, 2).

Foram feitos os 6° e 7° grupo de testes, onde no 6° grupo de testes foi escolhida uma situação onde é possível encontrar uma rota sem colisão, e no 7° grupo de testes uma situação onde se sabe que acontecerá uma colisão inevitavelmente com os alvos definidos.

Para estes testes se utilizará os alvos como definidos na Tab. 4.4, representados nas Fig. 4.10 e 4.11 que reconstroem as trajetórias sobrepostas dos dois braços, quando realizadas separadamente, e mostram as condições iniciais e finais dos braços. Nota-se na Fig. 4.11 que o braço 1 está sobre o braço 2 na condição final.

Tabela 4.4 – Testes escolhidos para avaliar o desempenho com as regras sobre o ganho

Grupo de testes	Alvo1 (px,py)	Alvo2 (px,py)
6	(1, 0)	(0.5, 1)
7	(2, 0)	(1.5, 2)

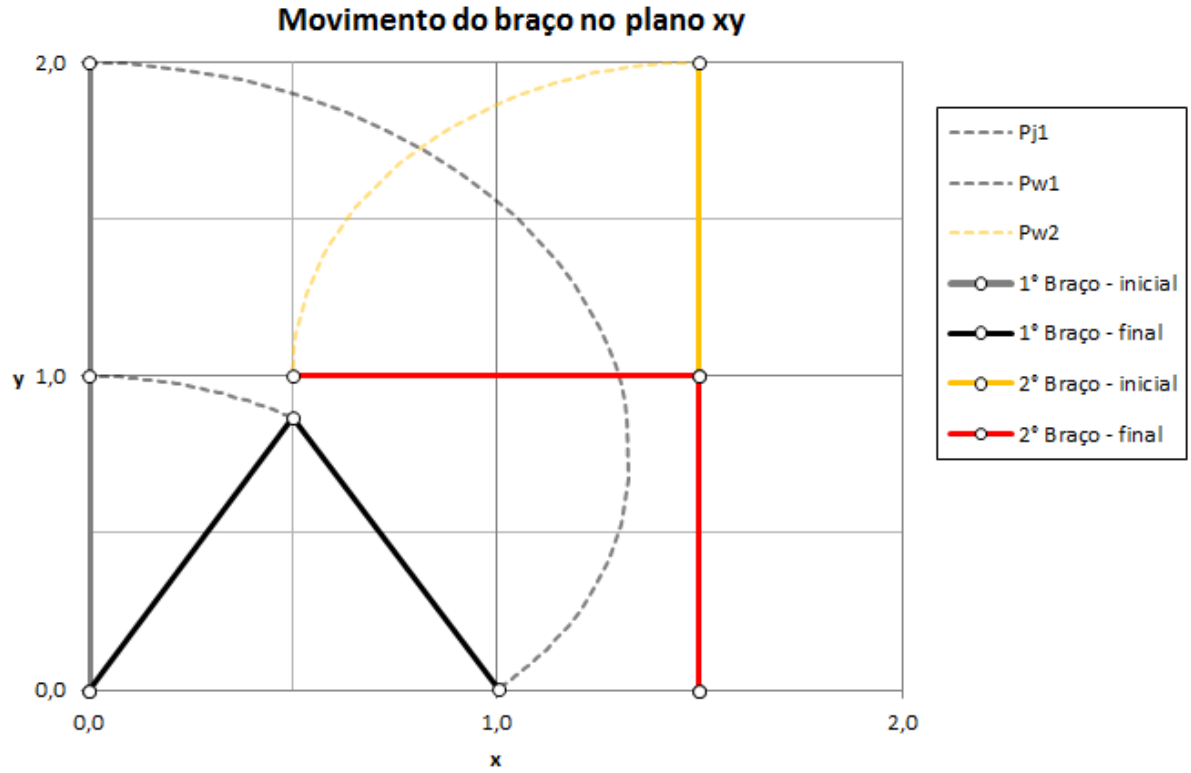

Figura 4.10 – Trajetória sobreposta dos braços para o 6° grupo de testes com a representação do braço nas condições inicial e final.

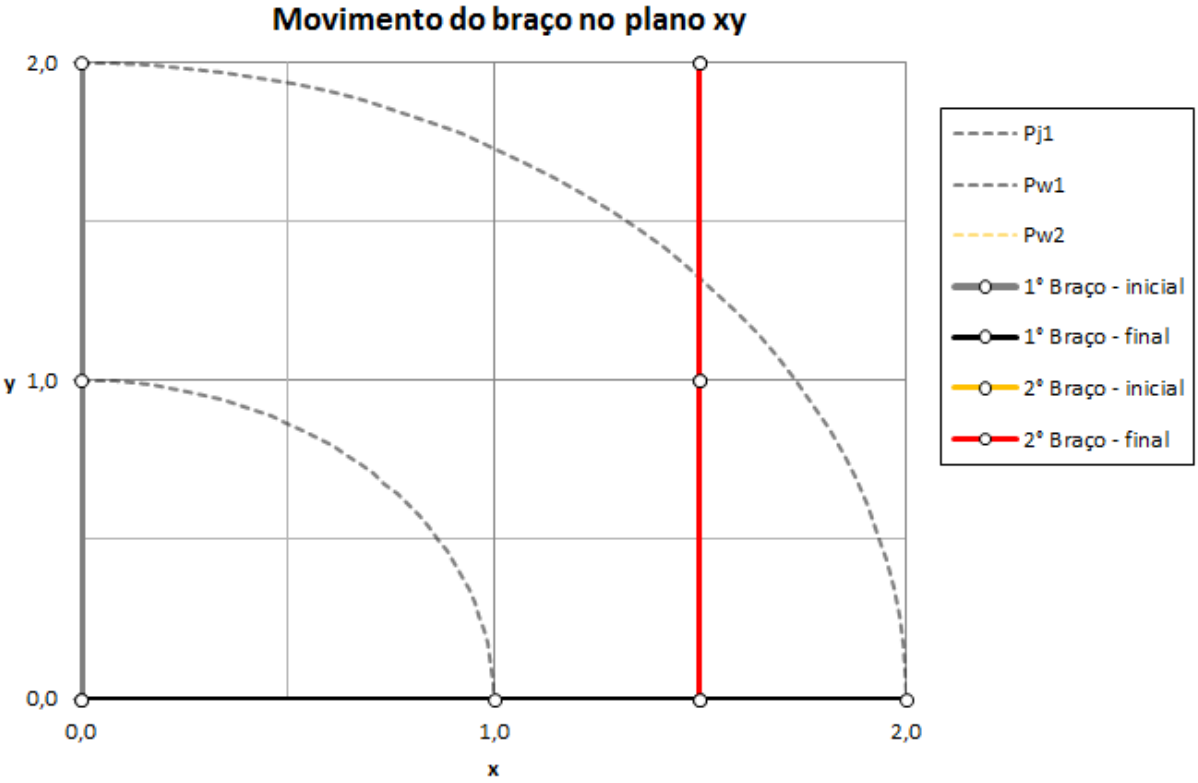

Figura 4.11 – Trajetória sobreposta dos braços para o 7° grupo de testes com a representação do braço nas condições inicial e final.

5. REALIZAÇÃO DOS TESTES EXPERIMENTAIS E DISCUSSÕES

5.1. 1° grupo de testes

Inicialmente, como descrito anteriormente, foram feitas simulações sem a presença do controle adicional inserido sistema embarcado, apenas o controlador projetado no sistema contínuo. Assim, a Figura 5.1 mostra a evolução do movimento do braço no plano xy nestas condições, para um primeiro alvo de (0,2).

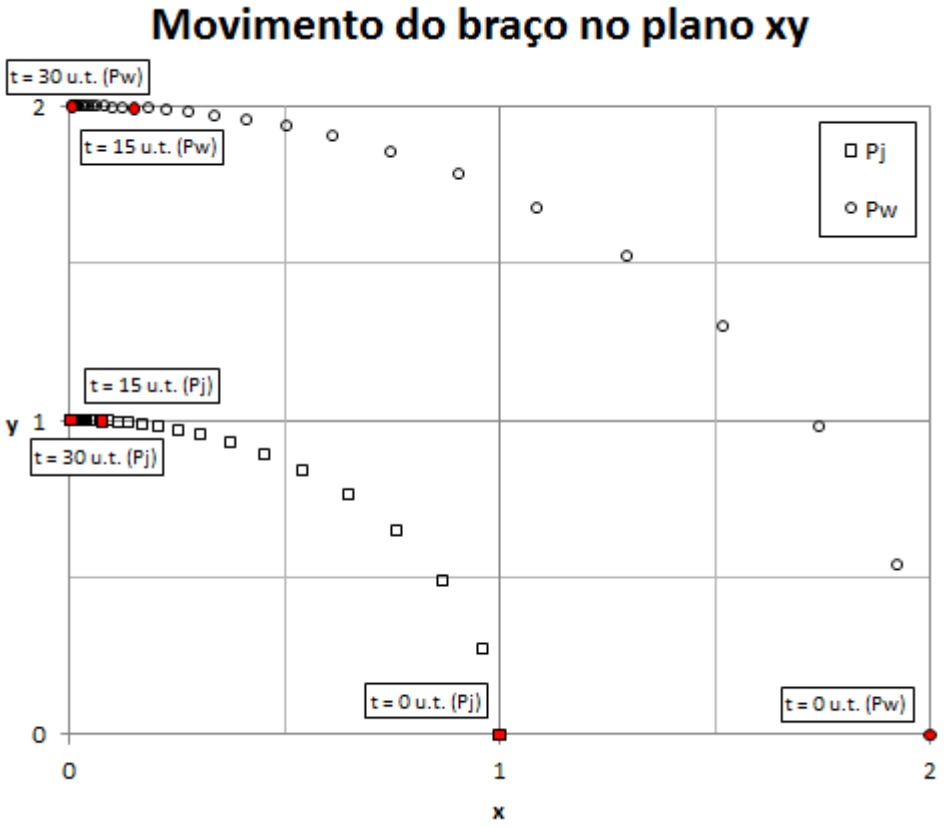

Figura 5.1 – Movimentação do braço ao longo do tempo, sem ação de controle, tendo ponto inicial do efetuador em (2, 0) e um alvo de (0, 2), com os instantes inicial, médio e final destacados em vermelho.

Para obter o tempo de estabilização, que pela Fig. 5.1 vê-se que é em torno de 20 u.t., precisa-se de um gráfico em função do tempo que é representado na Fig. 5.2, através distância normalizada ao alvo. Esta distância é calculada pela distância a cada instante do ponto Pw em relação às coordenadas do alvo e normalizada pela distância inicial entre estes pontos, de modo que se tem valores de 0 a 1. Nota-se pela Fig. 5.2. que o tempo de estabilização é cerca de 20 u.t.

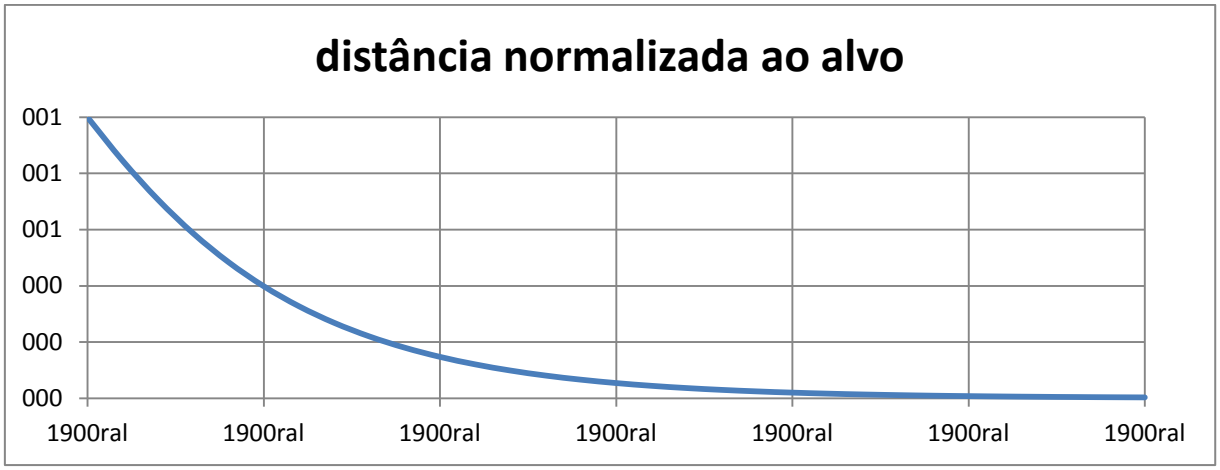

Figura 5.2 – Distância normalizada do efetuador ao alvo (0, 2), sem ação de controle.

Para verificação da robustez do controle embutido no modelo contínuo do sistema, verificou-se que este tempo não sofre grandes alterações quando é atribuído um alvo ao sistema que exige a movimentação independente dos dois motores. Isto é representado pela Fig. 5.3, que foi obtida com a co-simulação para o segundo alvo, de (0, 1), que apresenta na Fig. 5.4 sua respectiva a evolução no plano xy.

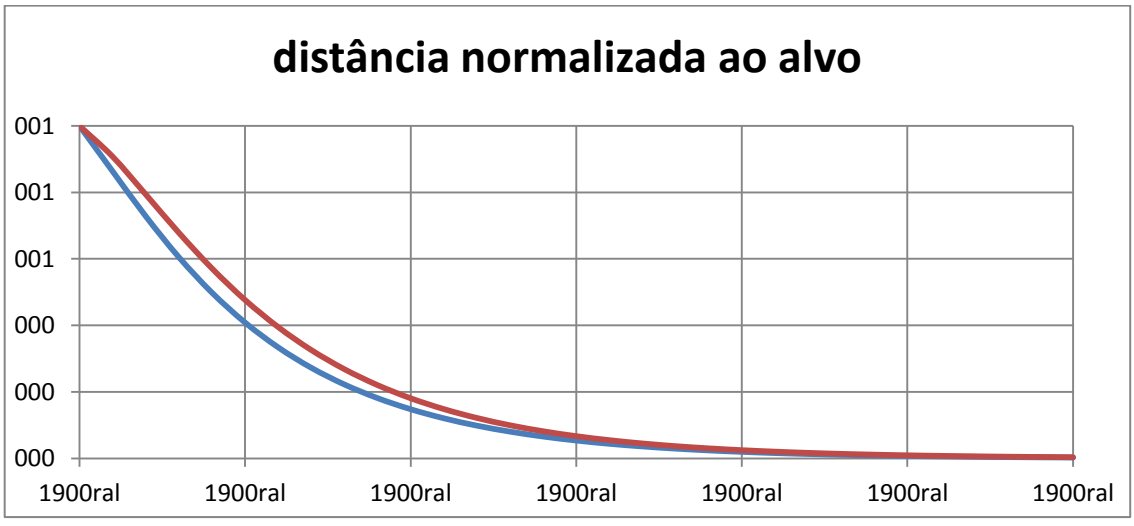

Figura 5.3 – Distância normalizada do efetuador ao alvo (0, 1), sem ação de controle.

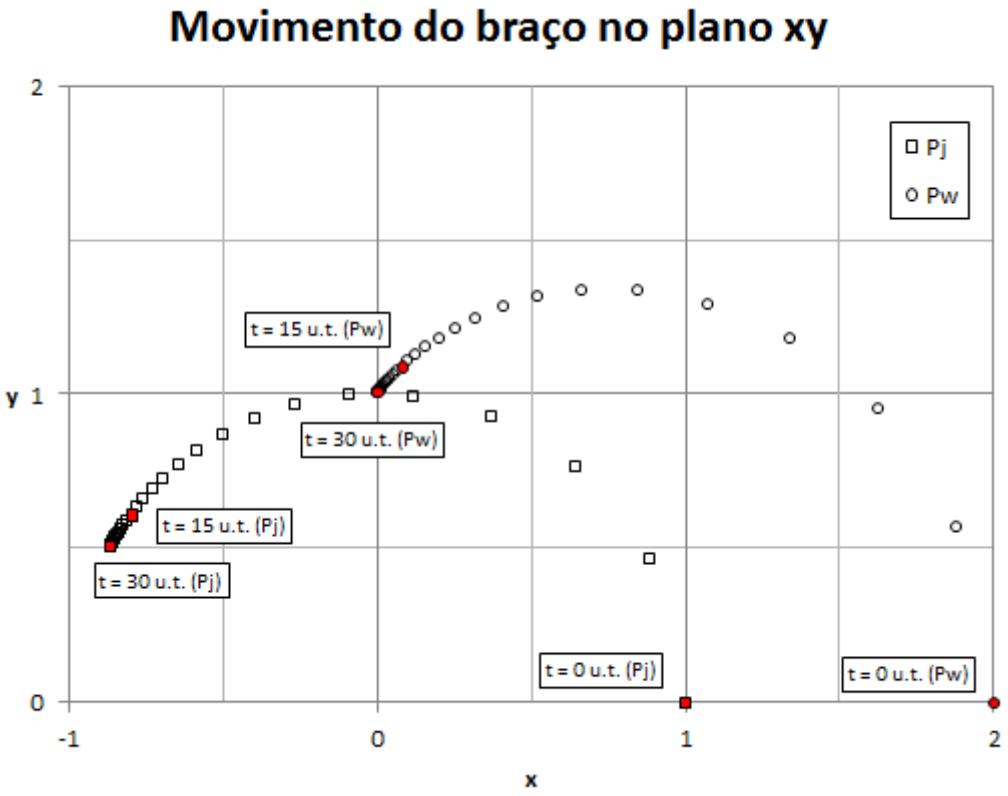

Figura 5.4 – Movimentação do braço ao longo do tempo, sem ação de controle, tendo ponto inicial do efetuador em (2, 0) e um alvo de (0, 1), com os instantes inicial, médio e final destacados em vermelho.

Com esta simulação, nota-se como o controle é desacoplado, tendo em vista que neste último teste (o qual prevê a atuação do controle sobre ambas as variáveis) o sistema é capaz de ter praticamente o mesmo desempenho.

5.2. 2° grupo de testes

Para o teste o 1° alvo (2,0), baseado na Tab. 3, prevê-se que o ganho permaneça pequeno para o regulador da segunda junta, pois o ângulo deve permanecer em 0°, e em contrapartida sofra grandes alterações para a primeira, que tem incialmente um erro de 90°. Entretanto vê-se, na Fig. 5.5, que não é exatamente isso que ocorre com as regras definidas preliminarmente.

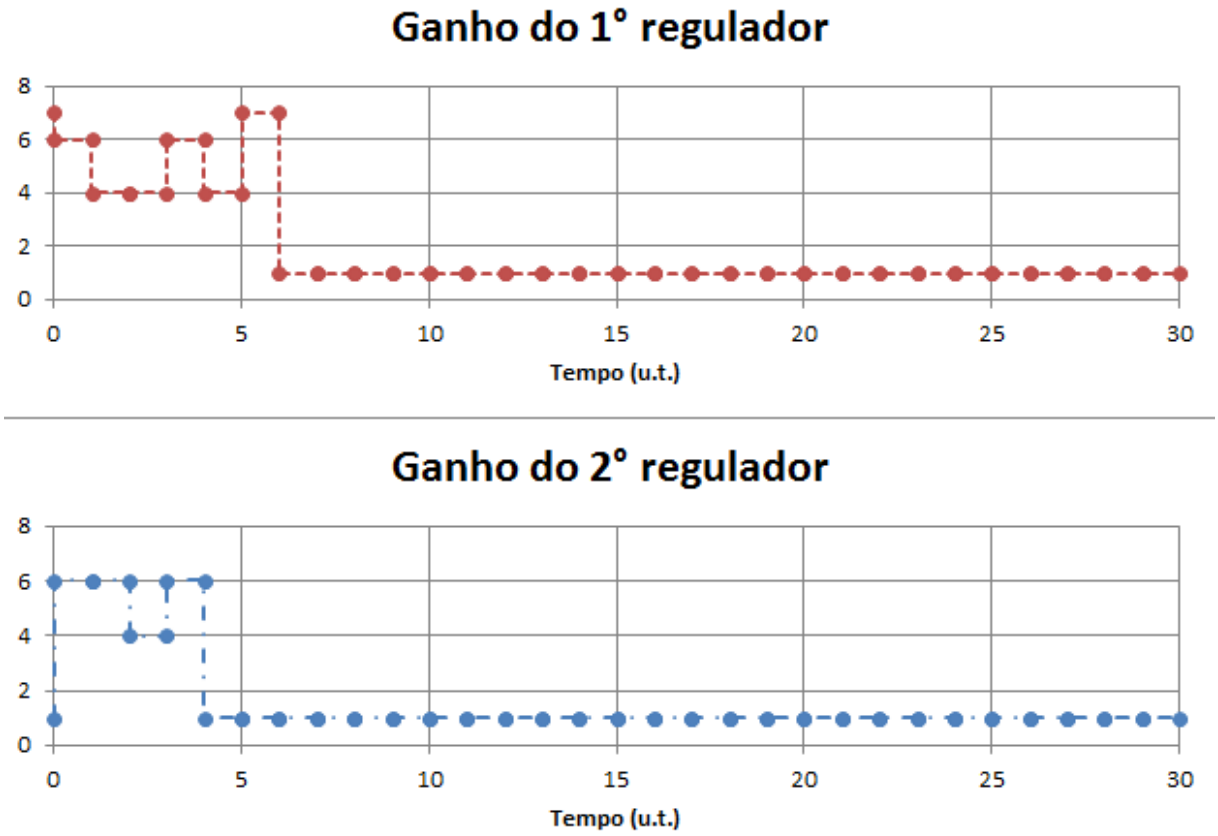

Figura 5.5 – Ganhos dos reguladores para o alvo em (0, 2) com apenas as regras preliminares de controle adaptativo.

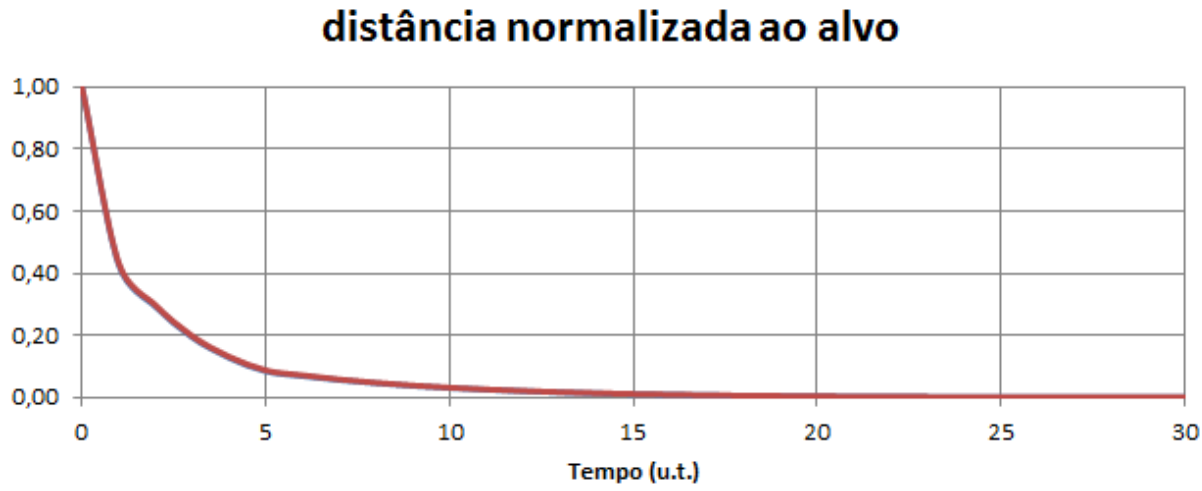

Figura 5.6 – Distância normalizada do efetuador ao alvo (0, 2), com apenas as regras preliminares de controle adaptativo.

Nota-se por outro lado que melhorou bastante a velocidade de resposta do sistema, com um tempo de estabilização de cerca de 10 u.t. Porém como mencionado, encontram-se resultados não esperados quanto aos ganhos.

Pelo gráfico do ganho do segundo regulador é visto que o ganho não permanece pequeno (na faixa entre 1 e 2) como deveria. Analisando a regra (2), reportada abaixo, conclui-se que deve ser devido ao tempo de amostragem do erro.

$$\text{Lasted}(e_i > err > e_j, T_{err} \rightarrow K_i < g < K_j) \qquad (2)$$

Como este foi definido em 5 u.t., nos primeiros 5 u.t., a regra (2) será sempre falsa e logo o sistema ainda não vê que o erro é pequeno, e logo escolhe um valor genérico entre 1 e 10, que é uma restrição global definida pela saturação do controlador.

Além disso, com o foco no gráfico do ganho do primeiro controlador, vê-se que não se obtém exatamente o que era esperado. Por exemplo, no instante 6 u.t. o ganho ainda é muito grande enquanto o erro é já muito pequeno.

Com esta percepção, propôs-se a alteração do valor de T_{err}, de modo que ele fosse grande apenas quando o tempo de amostragem fosse pequeno. Assim T_{err} é o arredondamento para cima do valor $1/\delta$ e consequentemente resolvem-se os dois problemas encontrados, pois desta forma o sistema é ajustado para mudar rapidamente os ganhos. Logo se obtêm os gráficos das Fig. 5.7, Fig. 5.8 e Fig. 5.9, com respectivamente a distância normalizada, os ganhos e a evolução no plano xy.

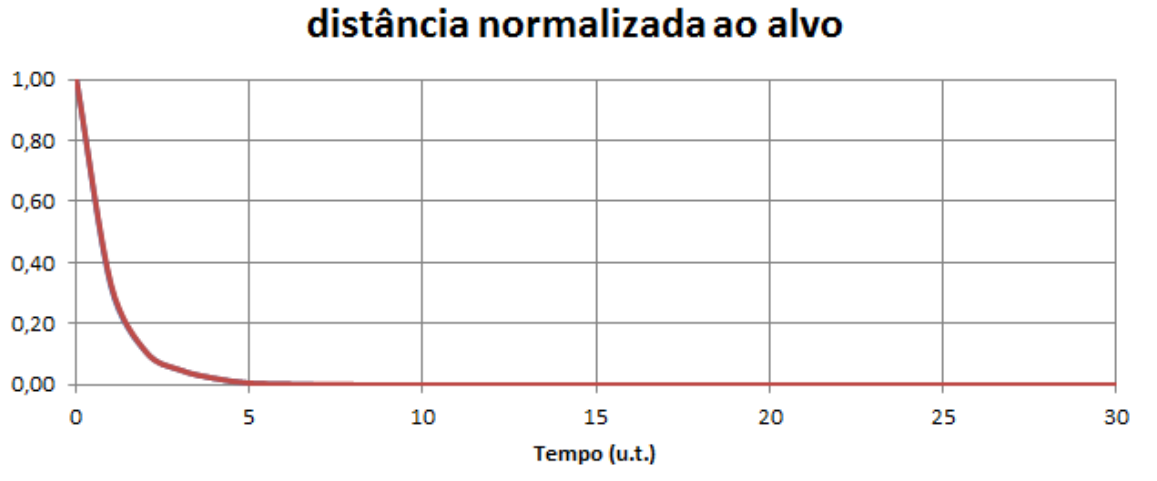

Figura 5.7 – Distância normalizada do efetuador ao alvo (0, 2), com o tempo de amostragem do erro corrigido.

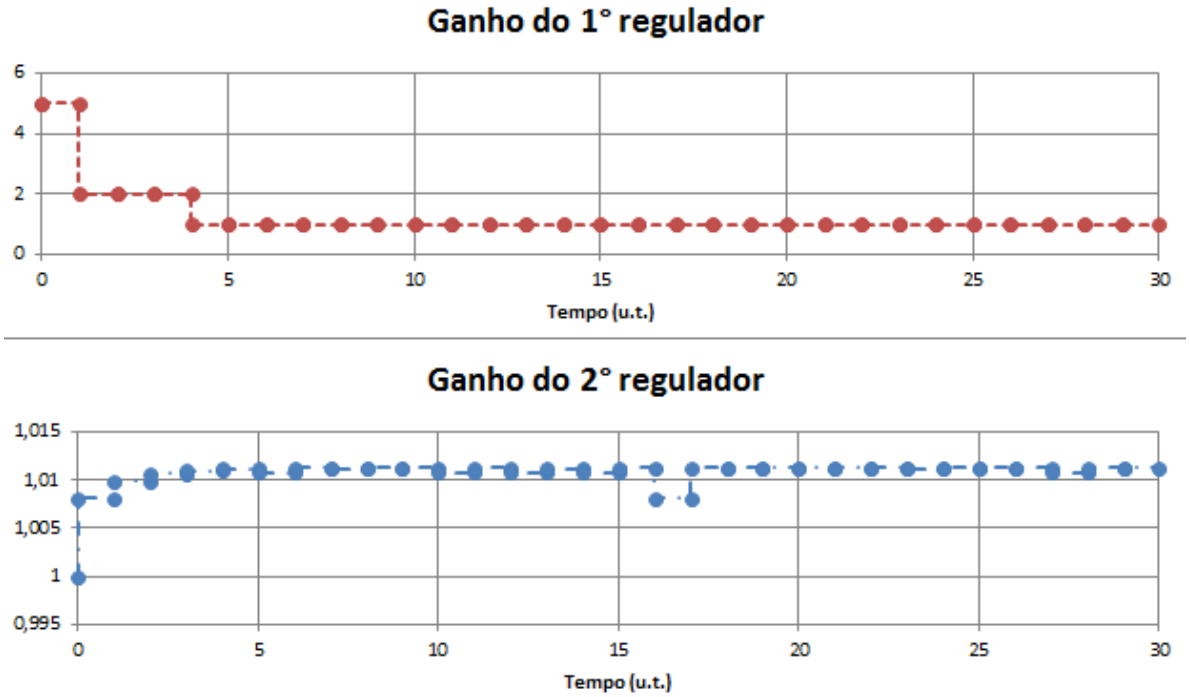

Figura 5.8 – Ganhos dos reguladores para o alvo em (0, 2), com o tempo de amostragem do erro corrigido.

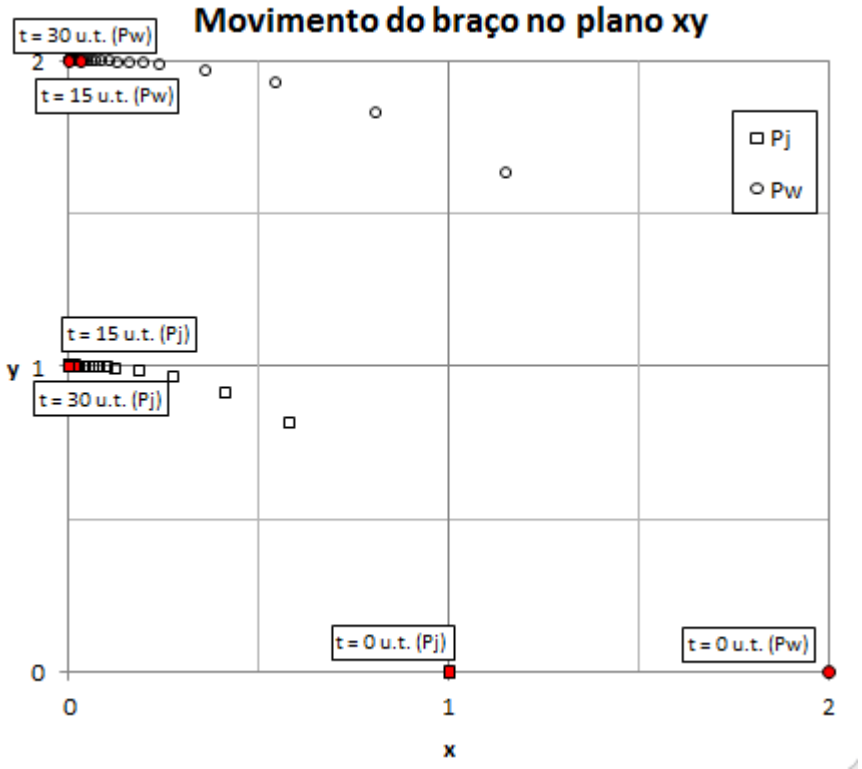

Figura 5.9 – Movimentação do braço ao longo do tempo, com o controle adaptativo, tendo ponto inicial do efetuador em (2, 0) e um alvo de (0, 2), com os instantes inicial, médio e final destacados em vermelho.

Com a correção do tempo de amostragem, o tempo de estabilização reduziu-se para 5 u.t. e os ganhos estão dentro dos intervalos pré-definidos. Nota-se a grande melhora que este controle traz ao sistema, com uma melhora de cerca de 83% no tempo de estabilização.

Por fim, realiza-se o teste com o 2° alvo (1, 0), para verificar que os controladores são capazes de trabalhar desacoplados, com os resultados reportados nas Figs 5.10, 5.11 e 5.12.

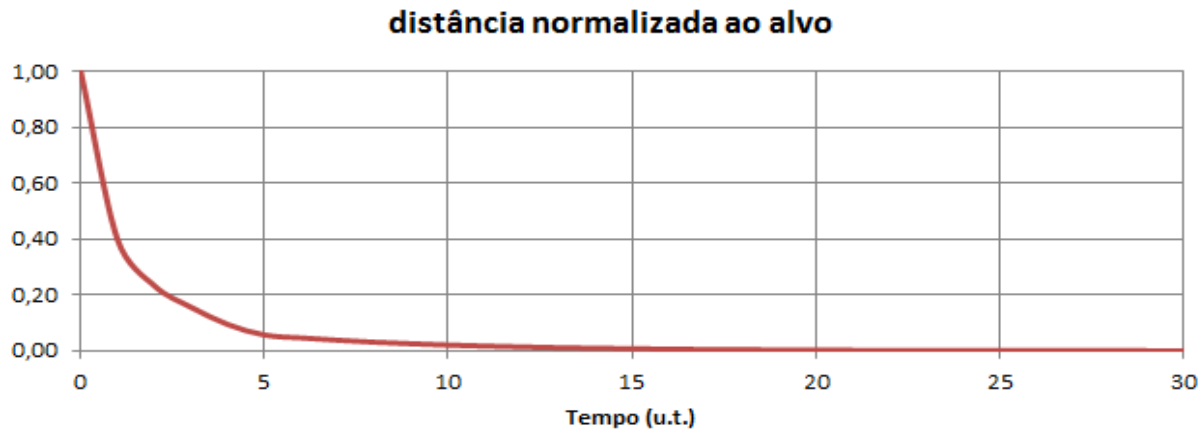

Figura 5.10 – Distância normalizada do efetuador ao alvo (0, 1), com o controle adaptativo.

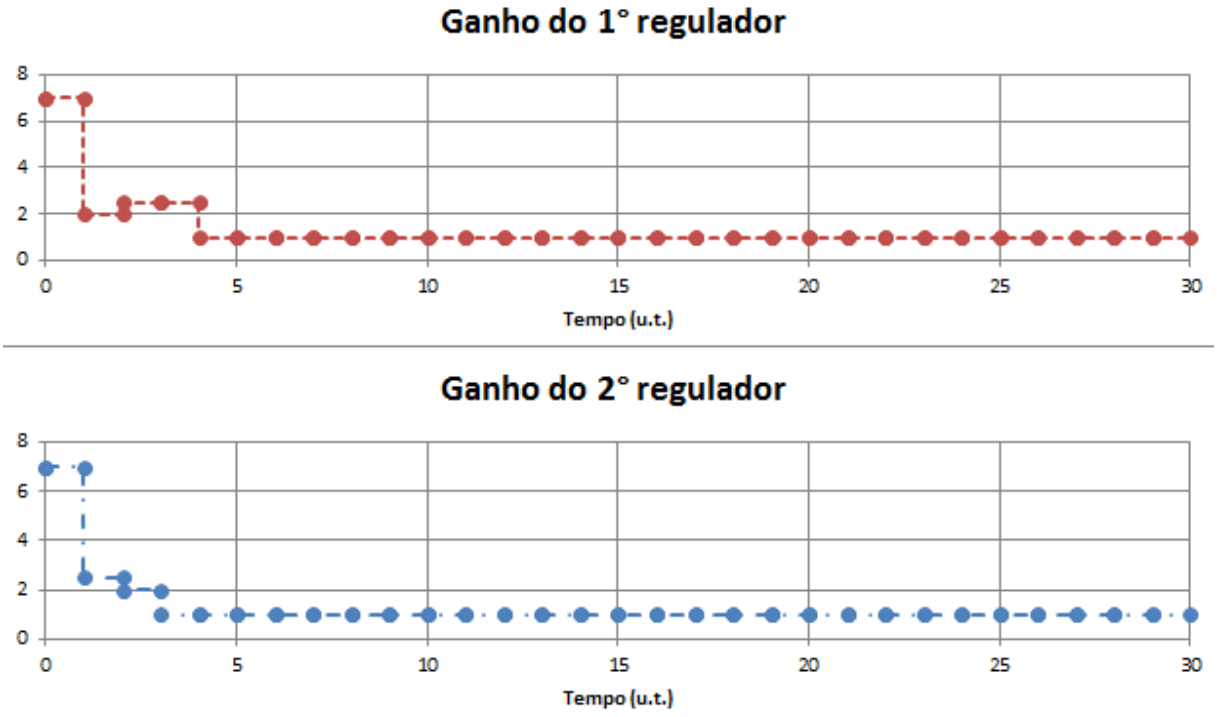

Figura 5.11 – Ganhos dos reguladores para o alvo em (0, 1), com o tempo de amostragem do erro corrigido.

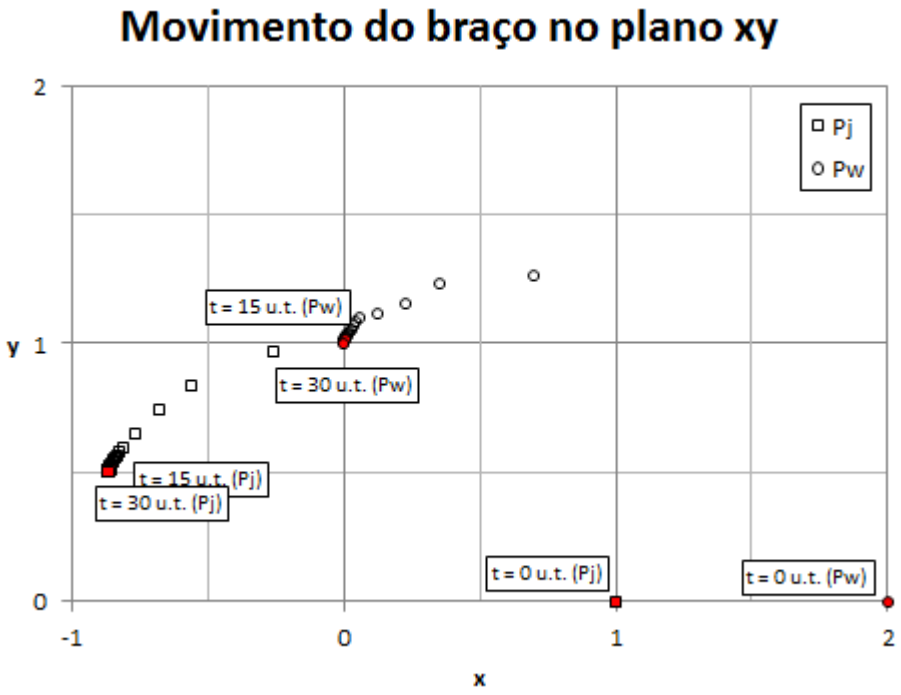

Figura 5.12 – Movimentação do braço ao longo do tempo, com o controle adaptativo, tendo ponto inicial do efetuador em (2, 0) e um alvo de (0, 1), com os instantes inicial, médio e final destacados em vermelho.

Dos gráficos vê-se que o tempo de estabilização tem uma pequena piora, mas ainda se mantém baixo (cerca de 7 u.t.) e os ganhos continuam respeitando as restrições pré-definidas.

Por fim, esta simulação confirma que os controles dos dois ângulos são desacoplados e, portanto concluiu-se a não necessidade da realização de testes para mais de um alvo nos testes sucessivos.

5.3. 3° grupo de testes

Inicialmente se reproduzem os gráficos dos ganhos para ver se o problema que exigiria a adição da regra (15) realmente ocorre (Fig. 5.13).

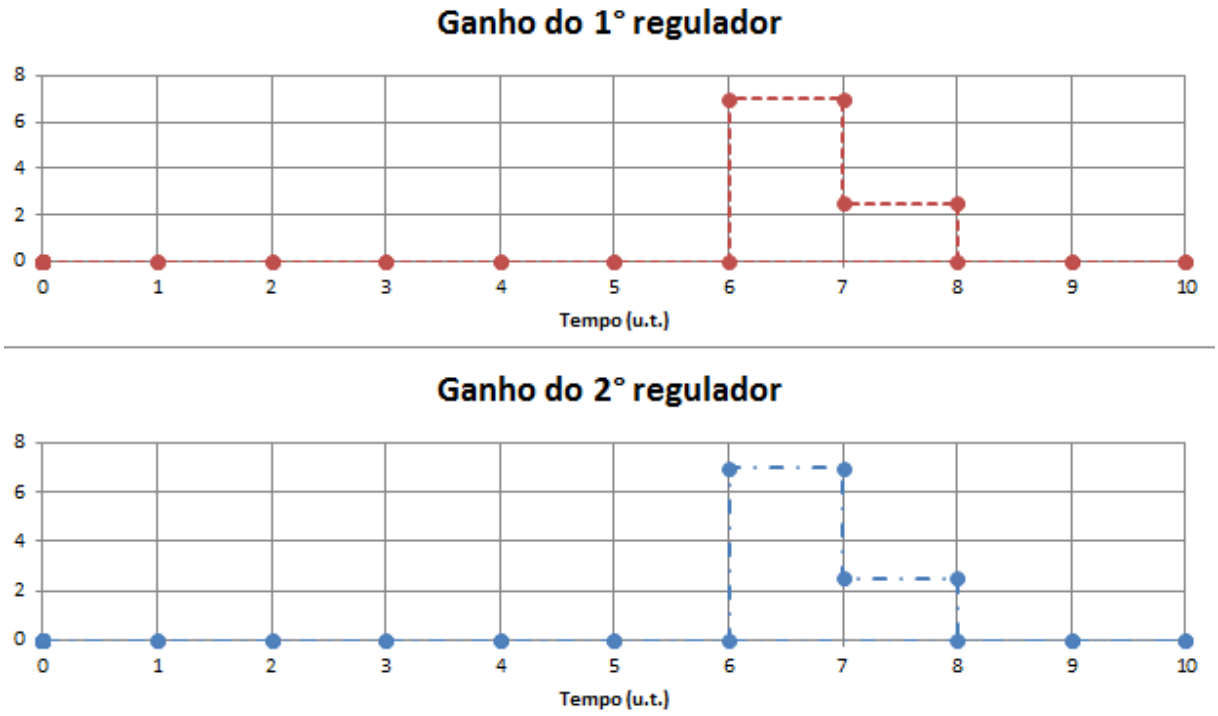

Figura 5.13 – Ganhos dos reguladores para o alvo em (0, 1), no cenário 2 sem a adição da regra (15).

Como nota-se pelo gráfico, a regra "DontMove" realmente ocorre quando não devia e portanto deve-se incluir a regra (15) e tem-se os seguintes resultados (Figs. 5.14, 5.15 e 5.16).

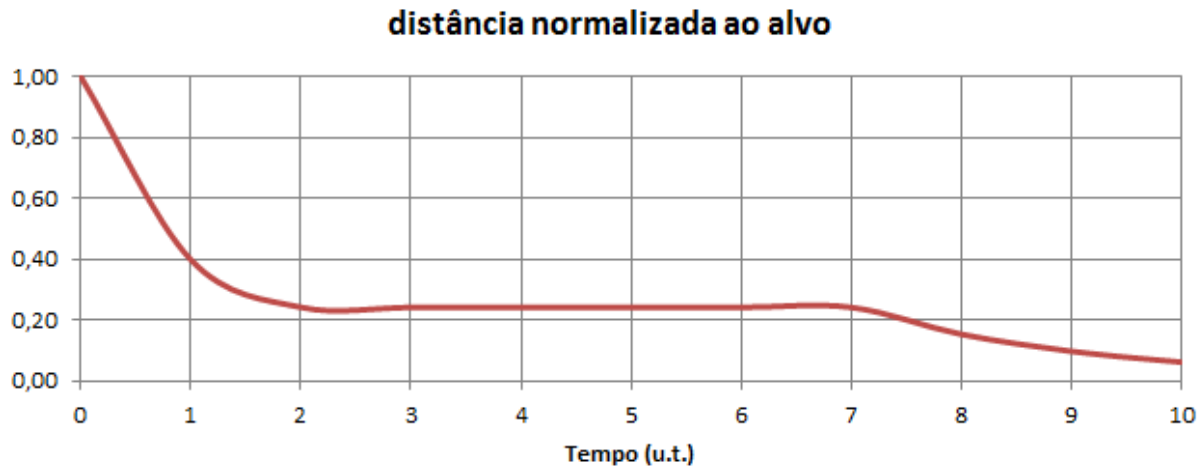

Figura 5.14 – Distância normalizada do efetuador ao alvo (0, 1), no cenário 2.

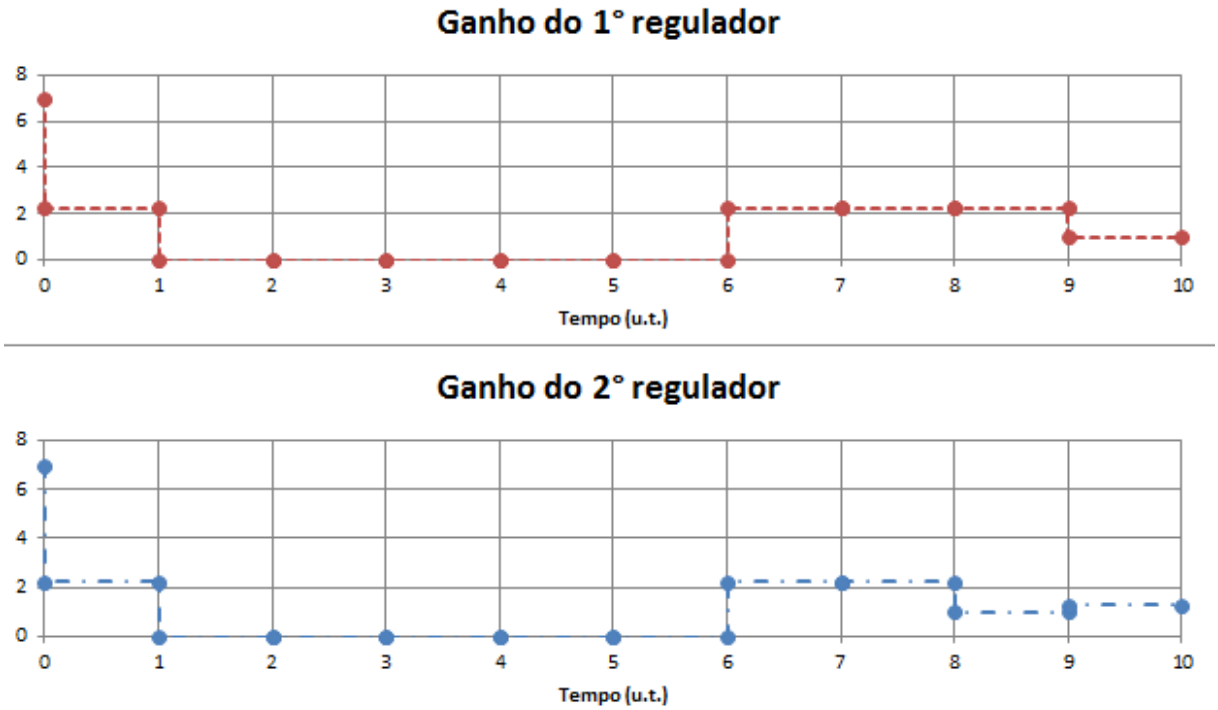

Figura 5.15 – Ganhos dos reguladores para o alvo em (0, 1), no cenário 2.

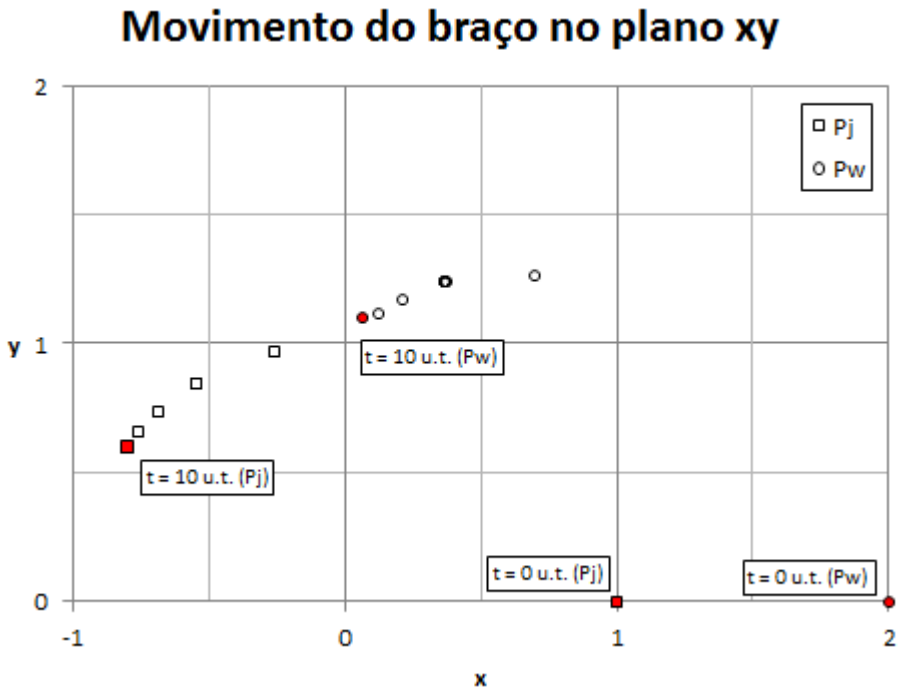

Figura 5.16 – Movimentação do braço ao longo do tempo, no 2° cenário, tendo ponto inicial do efetuador em (2, 0) e um alvo de (0, 1), com os instantes inicial e final destacados em vermelho.

A partir dos gráficos, vê-se que o resultado é consistente com o que era esperado, uma vez que os ganhos são anulados entre 1 u.t. e 6 u.t., e o sistema permanece inativo/parado durante 5 u.t. instantes.

5.4. 4° grupo de testes

Como mencionado anteriormente, é necessário comparar estes resultados com um teste relativo à regra que define uma forma alternativa de parar o braço, de acordo com as regras (10) e (11).

Para saber se a regra está funcionando corretamente, não adianta olhar para os gráficos dos ganhos, pois o controle não atua mais sobre eles, mas sim sobre os *Setpoints*. Portanto serão apresentados os gráficos *Setpoints* (Fig. 5.17).

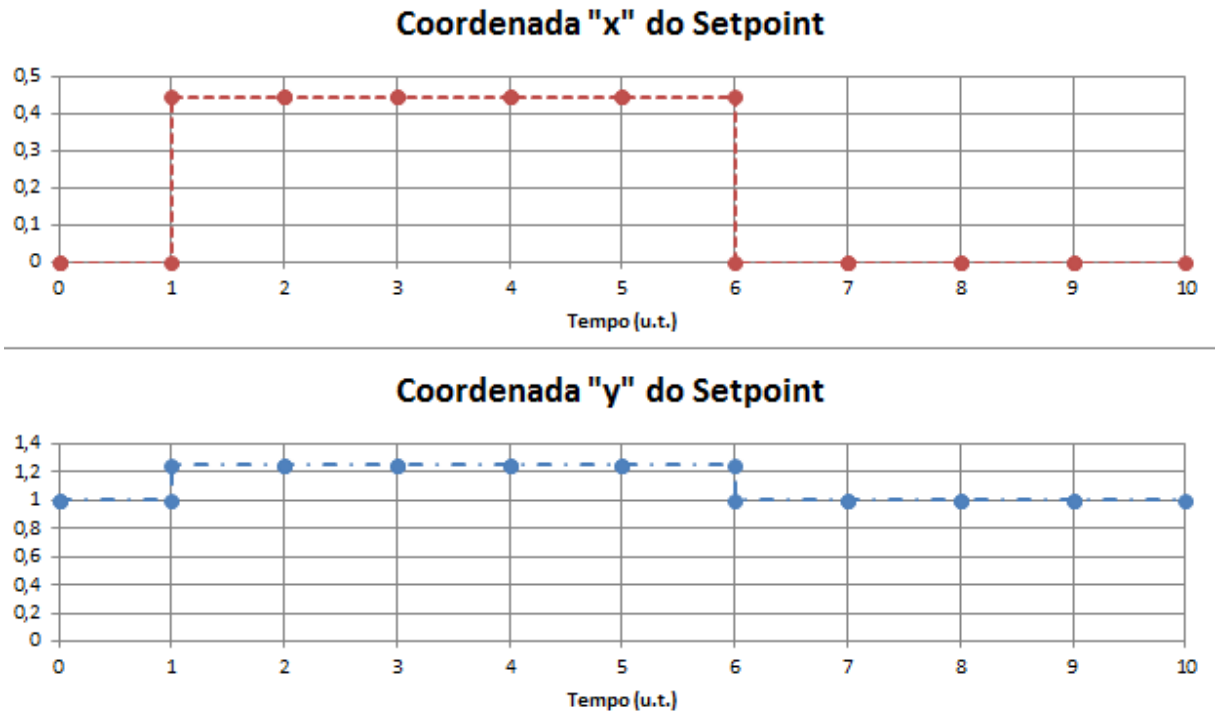

Figura 5.17 – *Setpoint* do sistema para o alvo em (0, 1), utilizando as regras (10) e (11) para determinar a parada do braço.

Vê-se no gráfico da Fig. 5.17 que a regra funciona como o esperado, com os reguladores recebendo um *Setpoint* diferente entre os instantes 1 u.t. e 6 u.t., e analisando o gráfico da Fig. 5.18, que apresenta as distâncias normalizadas para as duas possibilidades de definição da regra "DontMove", nota-se que os resultados estão sobrepostos no gráfico e logo as regras obtêm os mesmos resultados.

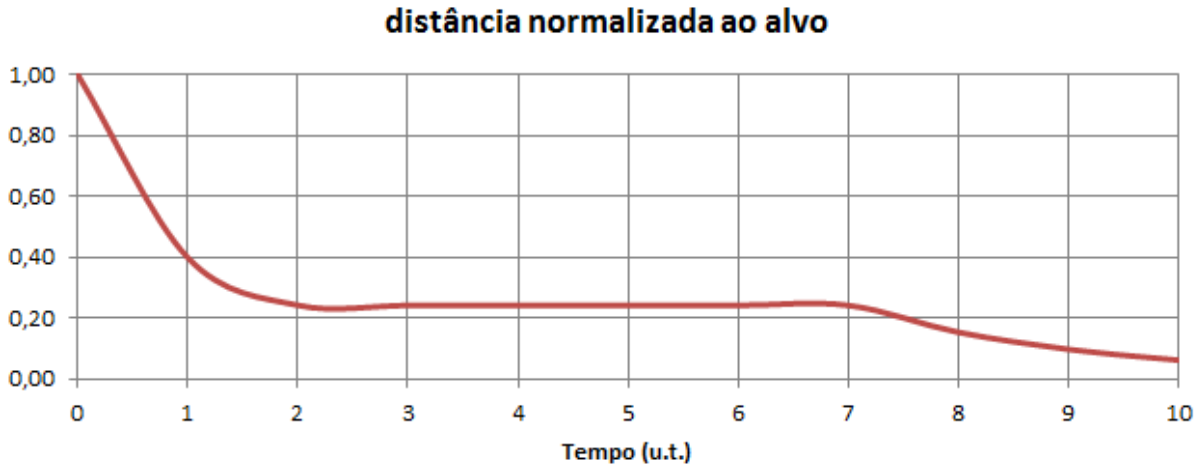

Figura 5.18 – Distâncias normalizadas do efetuador ao alvo (0, 1), no cenário 2, com a regra (9) em azul e as regras (10) e (11) em vermelho.

Dada a paridade de resultados obtidos encontra-se uma desvantagem em utilizar a primeira regra (9), a qual é a mais fácil de aplicar na simulação. Levar a zero um ganho de um regulador, na verdade, é uma prática geralmente não recomendável, pois pode gerar oscilações indesejadas ao sistema quando o controlador estiver sujeito a distúrbios.

Por outro lado, a segunda regra apresenta a vantagem de permitir adotar diferentes políticas ao detectar um obstáculo, ao invés de simplesmente bloquear o braço. O braço pode, por exemplo, fazer pequenos movimentos, ou ainda se mover em outra direção.

Este aspecto servirá posteriormente para controlar os sistemas constituídos por múltiplos braços robóticos. E como consequência nos próximos testes será utilizada apenas as regras (10) e (11).

5.5. 5° grupo de testes

Nestes testes foram feitas co-simulações com um alvo inicial (1, 0) e uma mudança de alvo para (2, 0) no instante 10 u.t, considerando a posição inicial do braço como vertical, ou seja, Pw = (0, 2). O gráfico representado na Fig. 5.19 é das coordenadas do efetuador, separadas em x e y de modo a verificar que o braço ficou parado durante 5 u.t.

A regra não respeita estritamente os requisitos que foram definidos anteriormente. O que se deseja é que se o braço chegar ao alvo, então que permaneça parado por T_{Target} instantes, para poder, por exemplo, fechar o braço sobre um objeto, mas depois disto que esteja apto a iniciar o movimento pra um novo alvo solicitado.

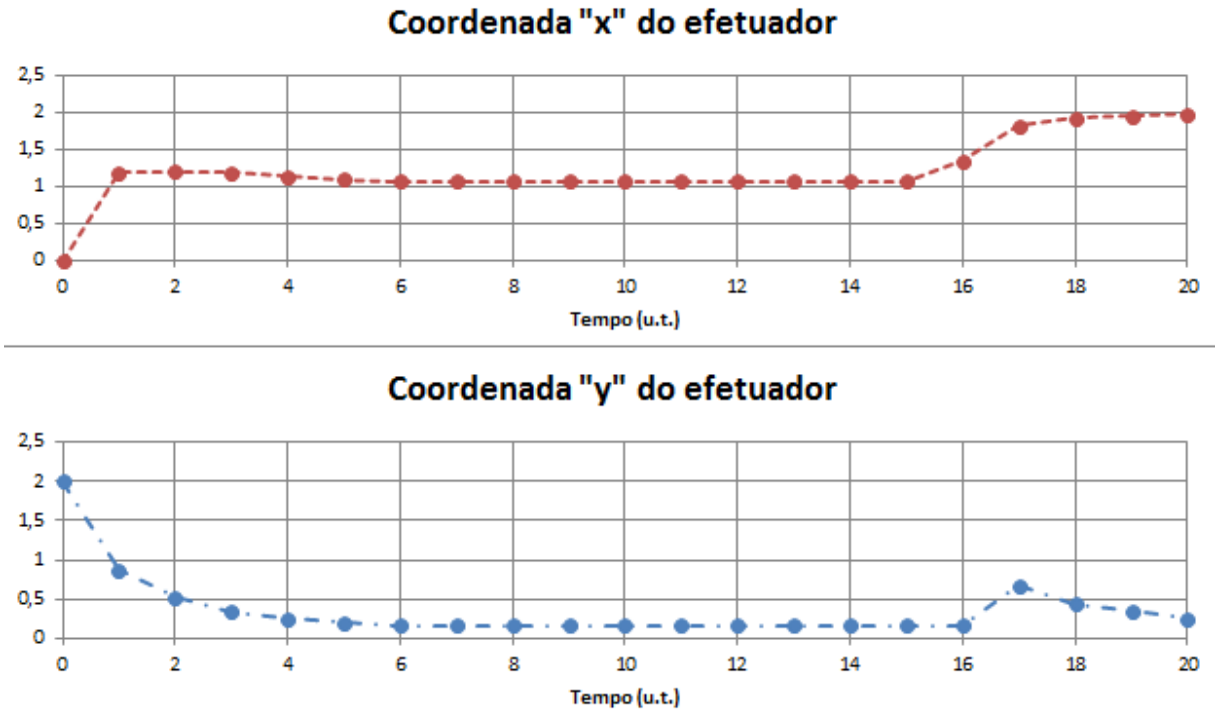

Figura 5.19 – Posição do efetuador decomposta nas coordenadas x e y em função do tempo, no cenário 3

No caso simulado, o sistema chega ao alvo no instante 5 u.t. e recebe um novo alvo no instante 10 u.t. Portanto, segundo as restrições definidas, ele deve ficar parado até o instante 14 u.t., e depois voltar a mover-se. Porém, o que ocorre é que o braço fica parado até o instante 20 u.t., porque a regra acaba por estabelecer que o sistema fique parado por 5 u.t. depois de uma mudança de alvo, e isto não é exatamente aquilo que se deseja.

O problema é que o parâmetro *OnTarget* permanece ativo até a mudança de Target, ou seja, de 5 u.t. a 10 u.t. e, portanto, satisfazendo a regra (4) até este instante, quando deveria satisfazê-la somente no instante 5 u.t.

Para resolver isso foi adicionada a regra (28) para fazer com que o sinal *OnTarget* gere o predicado somente no instante imediatamente após a chegada ao Target.

$$Alw\big(Past(\neg OnTarget, 1) \wedge OnTarget \rightarrow Lasts(DontMove\ , 5)\big) \qquad (28)$$

Com esta regra a mais, os novos resultados da simulação são apresentados na Fig. 5.20, onde se vê que o sistema atende ao esperado.

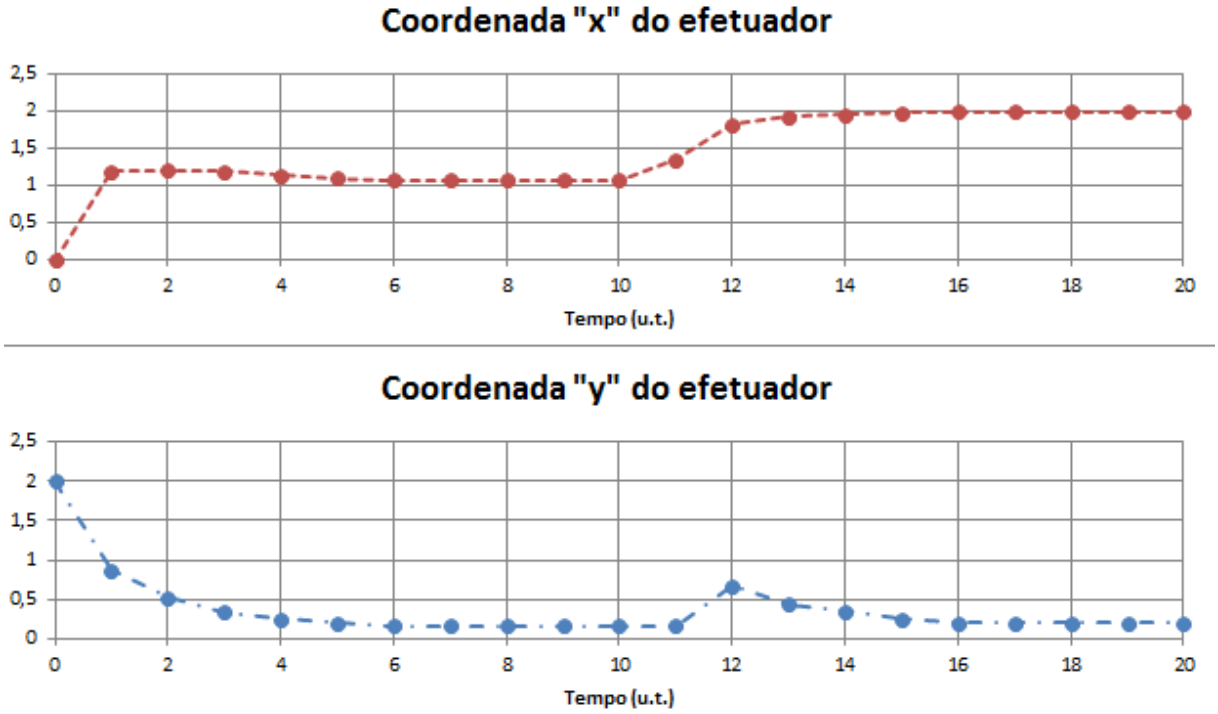

Figura 5.20 – Posição do efetuador decomposta nas coordenadas x e y em função do tempo, no cenário 3 com a regra (28).

5.6. 6° grupo de testes

Seguindo os procedimentos descritos anteriormente para o cálculo das distâncias, obtêm-se os resultados da Fig. 5.21, quando não aplicadas ainda as regras para evitar a colisão.

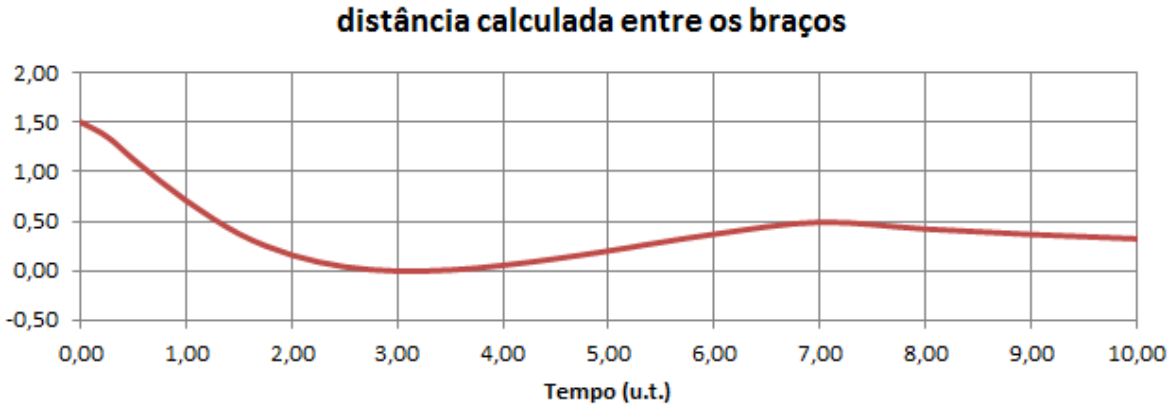

Figura 5.21 – Distância calculada entre os braços para o 6° grupo de testes sem o uso da política para evitar colisões.

Nota-se que em torno de 3 u.t. a distância se anula, indicando que ocorre uma colisão. Quando se implanta a política descrita anteriormente para o 4° cenário, adotando como dis-

tância limite o valor de 0.2 e crítica o valor de 0.1, tem-se como resultado o gráfico da Fig. 5.22, onde se nota que a distância não chega mais ao valor nulo.

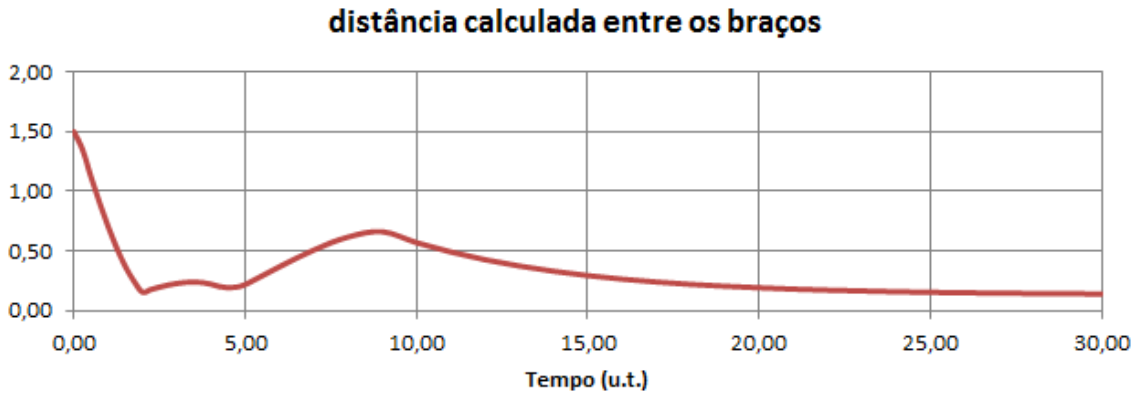

Figura 5.22 – Distância calculada entre os braços para o 6° grupo de testes com o uso da política para evitar colisões.

Porém para verificar se o sistema chegou ao seu alvo, deve-se representar a evolução do movimento, que foi separada em três etapas para melhor visualização (Fig. 5.23 a 5.25):

1. Movimento até a entrada na zona limite;
2. Deste instante até a saída da zona limite e volta a condição normal;
3. Deste instante até os braços chegarem ao alvo;

5.7. 7° grupo de testes

Analogamente ao 6° grupo de testes, inicialmente realiza-se um teste sem as regras de colisão, obtendo os resultados da Fig. 5.26.

Nota-se que em 4 u.t. a distância se anula, após isso se torna negativa, de modo a indicar que o braço “atravessou” o outro, o que é impossível fisicamente, mas este calculo baseia-se numa sobreposição de movimentos, logo faz sentido e ajuda a indicar as colisões.

Quando se implanta a politica descrita anteriormente para o 4° cenário, adotando os mesmos paramentos do 6° grupo de testes, tem-se como resultado o gráfico da Fig. 5.27, onde nota-se que novamente a distância não chega mais ao valor nulo e nem negativo.

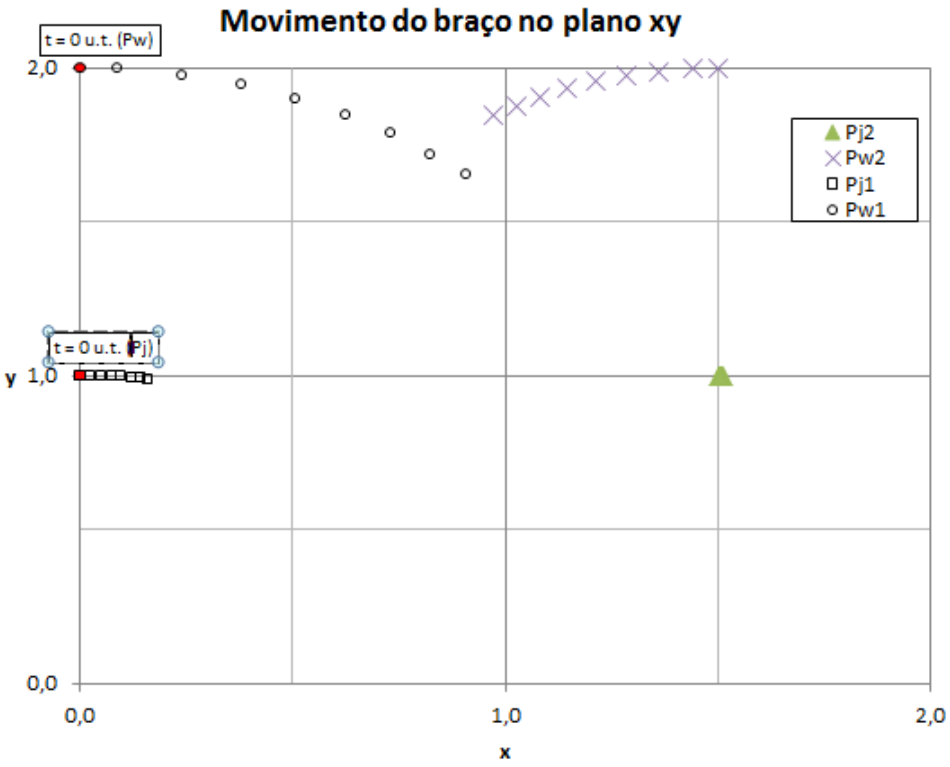

Figura 5.23 – Evolução do sistema, no 6° grupo de testes, do instante inicial até a entrada na zona limite.

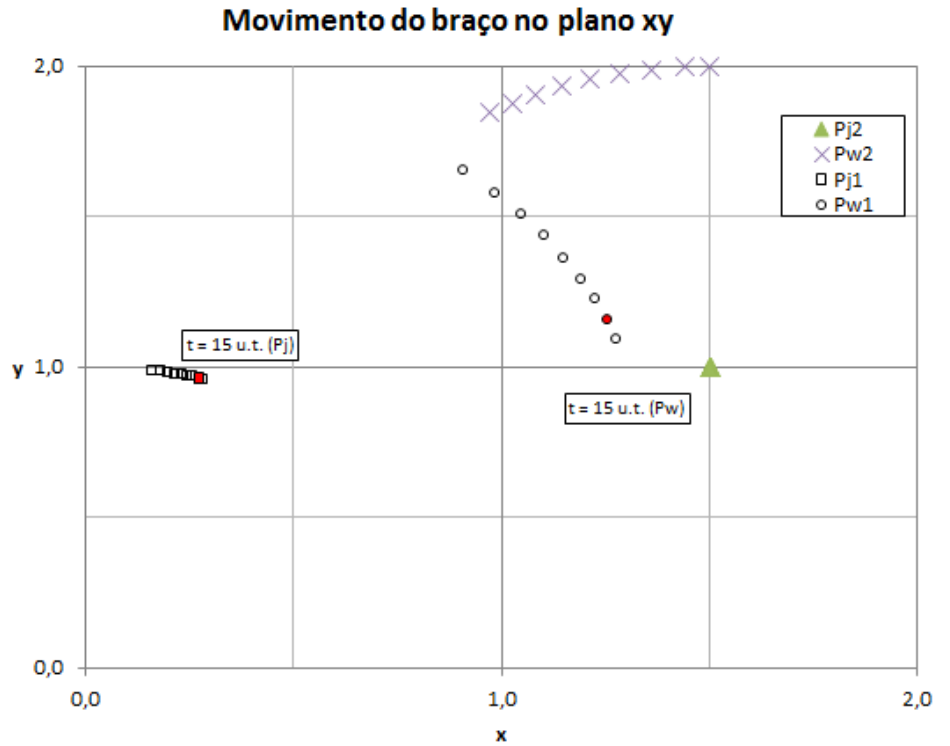

Figura 5.24 – Evolução do sistema, no 6° grupo de testes, da entrada na zona limite até a volta a situação normal.

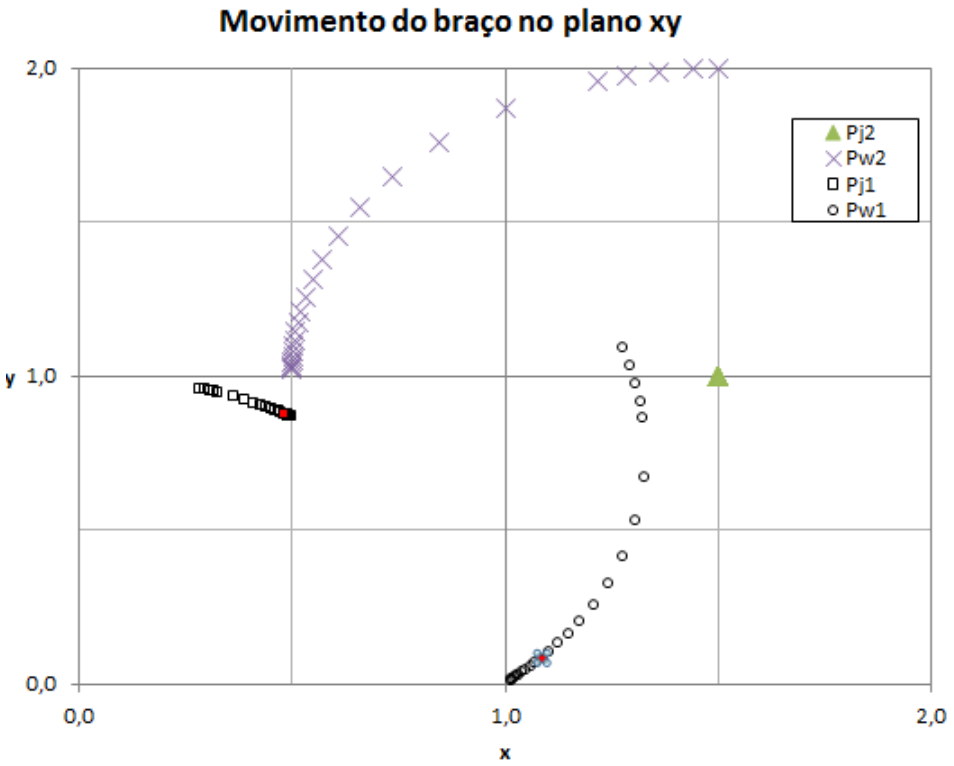

Figura 5.25 – Evolução do sistema, no 6° grupo de testes, da volta a situação normal até o 1° braço chegar ao alvo.

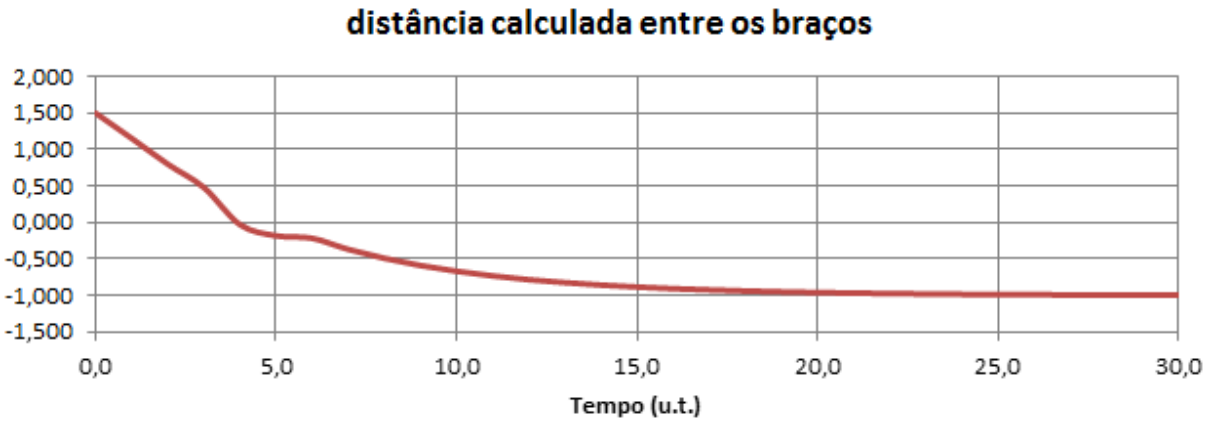

Figura 5.26 – Distância calculada entre os braços para o 7° grupo de testes sem o uso da política para evitar colisões.

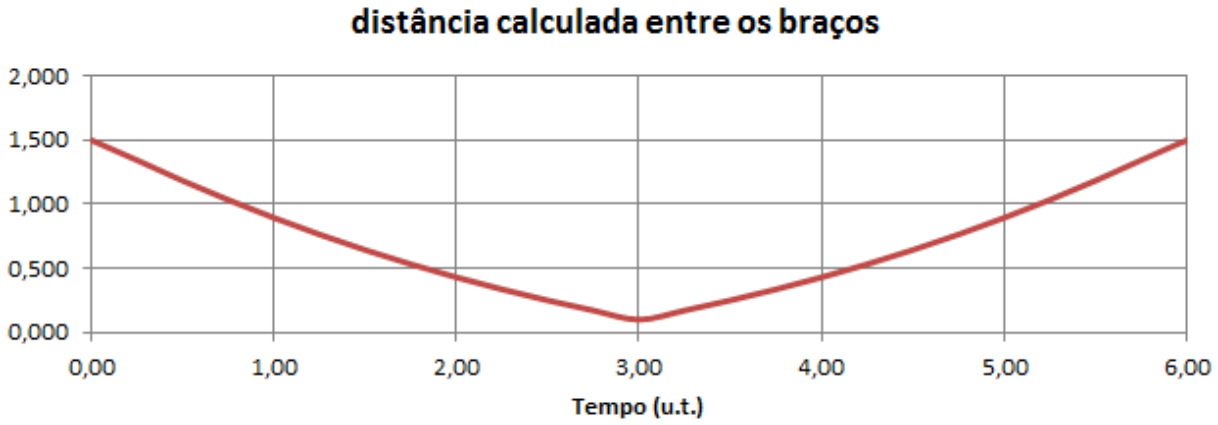

Figura 5.27 – Distância calculada entre os braços para o 7° grupo de testes com o uso da política para evitar colisões.

Ao representar a evolução do movimento (Fig. 5.28 a 5.29), separada em duas etapas para melhor visualização (Movimento até a entrada na zona crítica, e volta à situação inicial) nota-se que, como esperado, o sistema não consegue chegar ao seu alvo.

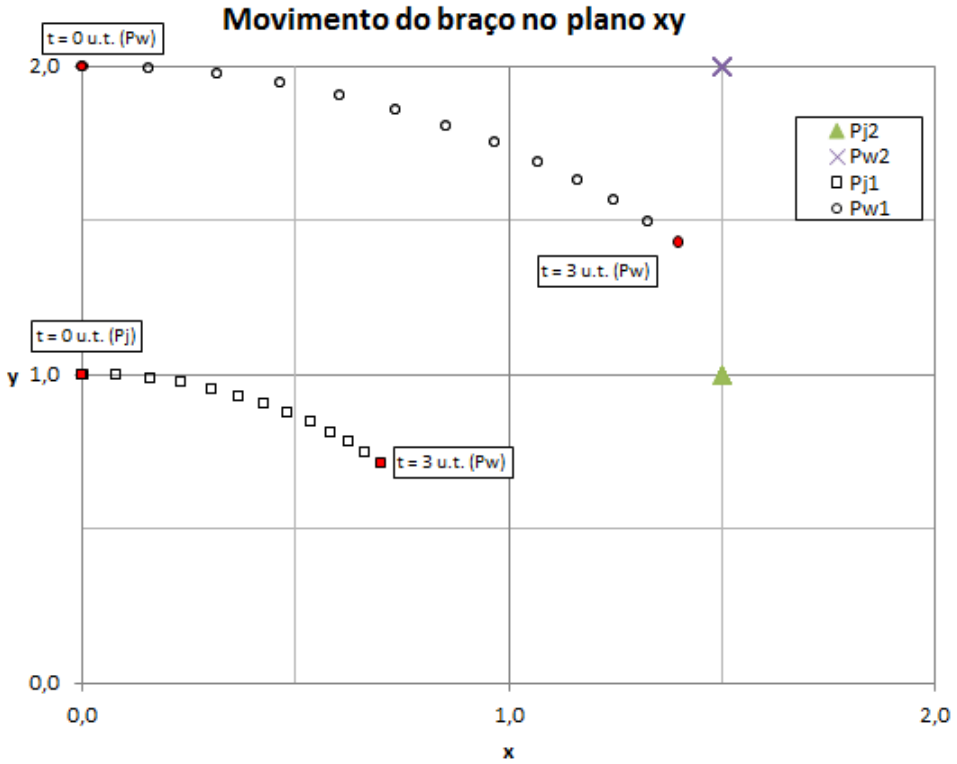

Figura 5.28 – Evolução do sistema, no 7° grupo de testes, do instante inicial até a entrada na zona crítica.

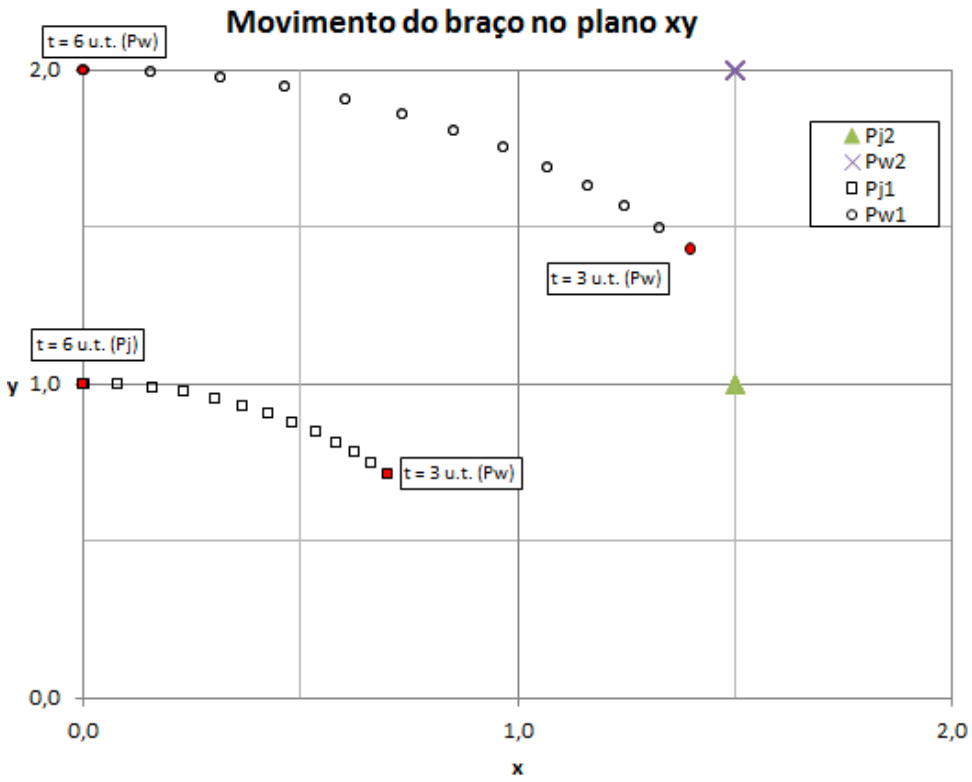

Figura 5.29 – Evolução do sistema, no 7° grupo de testes, da entrada na zona crítica até a volta a situação inicial.

6. CONCLUSÕES

Sistemas cada vez mais complexos requerem a modelagem completa destes em um ambiente virtual, e o método da co-simulação mostrou-se uma alternativa válida, pois elimina a necessidade da obtenção de um modelo único, que requer complexas derivações analíticas, e que dada a crescente complexidade dos sistemas, está se tornando cada vez mais difícil.

Por meio desta abordagem, permite-se ainda adotar abordagens ótimas diferentes para modelagem de cada parte do sistema, podendo muitas vezes testá-las separadamente, deixando a cargo do co-simulador o papel de integrá-las e obter a evolução completa do sistema. A ferramenta de co-simulação adotada, o MCA (*MADES Cossimulation Approach*), que possui como principal característica ser não determinística, deixando certo grau de liberdade ao sistema embarcado para escolher a melhor solução, o que permite ao sistema adaptar-se a distúrbios e pequenas modificações do meio.

Tal abordagem sugere o uso do Modelica para o ambiente que por ser não causal e orientado a objetos, permitiu um projeto mais visual do sistema, tornando-o mais intuitivo. Do outro lado usou-se o Zot que define o sistema embarcado através da definição de regras lógicas, o que se mostrou de fácil utilização, pois os vínculos são definidos de maneira muito próxima de como são ditos verbalmente.

Pra aumentar a repetibilidade, a fim que outras pessoas possam usar este co-simulador (MCA) em vários outros exemplos de aplicação, propôs-se uma sistemática de projeto que permite a aplicação da co-simulação.

Foi então aplicada esta sistemática a um estudo de caso, com a modelagem dos braços robóticos, motores e controladores em OpenModelica, e para diversos cenários foi escrito um conjunto de regras, que foram então testadas e refinadas. Notou-se que uma vez que modelada e bem definida a planta, pode-se focar no projeto do sistema embarcado, focando na consistência das regras (definidas em TRIO e interpretadas pelo Zot), confrontando os resultados obtidos com aquilo que era esperado, corrigindo eventuais falhas e adicionando novos vínculos não pensados anteriormente, tornando o desenvolvimento “dinâmico”.

Por fim espera-se que este trabalho sirva de estímulo para que outras pessoas usem da co-simulação para modelagem de sistemas complexos, ou que adotem o OpenModelica ou o Zot para modelagem nas respectivas áreas que estes programas são focados. Além disso, deseja-se que no futuro seja possível usar desta abordagem para uma aplicação em tempo real,

substituindo o modelo do ambiente, passando a interagir com os sensores e atuadores do sistema físico.

REFERÊNCIAS BIBLIOGRÁFICAS

AL-HAMMOURI, A.; BRANICKY, M.; LIBERATORE, V., 2008. *Co-simulation for networked control systems*, in 2008 Hybrid Systems: Computation and Control (HSCC 2008), pp. 16–29.

ARZEN, K. E.; OLSSON, R.; AKESSON, J., 2002. *Grafchart for procedural operator support tasks*, in Proc. of the IFAC World Congress.

ÅSTRÖM, K. J., WITTENMARK, B., 1995. *Adaptive control*, 2nd edition, Addison-Wesley.

BAGNATO, A.; BARESI, L.; KOLOVOS, D.S.; MORZENTI, A.; PAIGE, R.F.; ROSSI, M.; SADOVYKH, A., 2010. MADES: *Embedded Systems Engineering Approach in the Avionics Domain*, in Proc. of HoPES 04 workshop at the European Conference on Modeling - Foundations and Applications (ECMFA 2010), Paris, France, June 2010.

BARESI, L.; FERRETTI, G.; LEVA, A.; ROSSI, M., 2012. *Flexible Logic-based Co-simulation of Modelica Models*, Dipartimento di Elettronica e Informazione - Politecnico di Milano.

BARESI, L.; MORZENTI, A.; MOTTA, A.; ROSSI, M., 2010. *Towards the uml-based formal verification of timed systems*, in Proc. of FMCO, ser. LNCS, vol. 6957, p. 267–286.

BASILE, F.; CACCAVALE, F.; CHIACCHIO, P.; COPPOLA, J.; MARINO, A., 2013. *A decentralized kinematic control architecture for collaborative and cooperative multi-arm systems*, in Mechatronics (2013), Pag.1-13 ISSN:0957-4158.

BELTRAME, T., 2006. *Design and development of a Dymola/Modelica library for discrete event-oriented systems using DEVS methodology*. ETH Zurich.

BERGERO, F.; KOFMAN, E., 2010. *Powerdevs: a tool for hybrid system modeling and real-time simulation*, SIMULATION.

BERNINI, G., 2009. *Logiche Temporali al servizio dell'Ingegneria del Software: CTL, CTL*, TRIO e LTLB*. Monografia, Mestrado em Engenharia do Software, Università degli Studi di Roma, Roma, Itália.

BROENINK, J.F.; DAMSTRA, A.S.; GROOTHUIS, M.A., 2008. *Virtual Prototyping through Co-simulation of a Cartesian Plotter*. In: 13th IEEE International Conference on Emerging Technologies and Factory Automation, 15-18 Sept 2008, Hamburg, Germany.

BUENO, R.C., 2013. *Simulazione ibrida di sistemi meccatronici.* Trabalho de conclusão de curso elaborado no setor Sistemi di elaborazione delle informazioni - Politecnico di Milano.

CELLIER, F.E., 1991. *Continuous System Modeling*, Springer - Verlag, New York, 1991

CHELLAS, B. F., 1980. *Modal Logic: An Introduction*. Cambridge University Press.

CIAPESSONI, E.; MIRANDOLA, P.; COEN-PORISINI, A.; MANDRIOLI, D.; MORZENTI, A., 1999. *From formal models to formally based methods: An industrial experience*, ACM TOSEM, vol. 8, no. 1, p. 79–113.

DAVID, R.; ALLA, H., 1992. *Petri Nets and Grafcet: Tools for Modelling Discrete Event Systems*. Prentice Hall.

FERRETTI, G.; GRITTI, M.; MAGNANI, G.; ROCCO, P., 2003. *A Remote User Interface to Modelica Robot Models*, Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy, pp. 231-240.

FERRETTI, G.; MAGNANI, G.; ROCCO, P., 2004.*Virtual prototyping of mechatronic systems*.IFAC Journal Annual Reviews in Control. Vol. 28, N° 2, 2004, pp. 193-206.

FRITZSON, P. A. et al., 2013. *OpenModelica Users Guide*, Open Source Modelica Consortium.

FRITZSON, P. A., 2004. *Principles of object-oriented modeling and simulation with Modelica 2.1.*, John Wiley and Sons.

FURIA, C. A.; ROSSI, M., 2010. *A theory of sampling for continuous-time metric temporal logic*, ACM TOCL, vol. 12, no. 1, pp. 8:1–8:40.

FURIA, C.A.; MANDRIOLI, D.; MORZENTI, A.; ROSSI, M., 2012. *Modeling Time in Computing*, XVI, 423 p., Cap 9.

HAREL, D., 1987. *Statecharts: A visual formalism for complex systems*, Sci. Comput. Program., vol. 8, p. 231–274.

HAREL, D.; PNUELI, A., 1985. *On the Development of Reactive Systems*, in Logics and Models of Concurrent Systems (K. R. Apt, ed.), NATO ASI Series, Vol. F-13, Springer-Verlag, New York.

HASLE, P. F. V.; OHRSTROM, P., 1995. *Temporal Logic: From Ancient Ideas to Artificial Intelligence*. Kluwer Academic Publishers.

HENZINGER, T. A.; HO, P.-H.; WONG-TOI, H., 1997. *HYTECH: A model checker for hybrid systems*, Int. J. on STTT, vol. 1, no. 1–2.

JERE, W., 2011. *Amesim*, International Book Marketing Service Ltd.

KARNOPP, D. C.; MARGOLIS, D. L.; ROSENBERG, R. C., 1990. *Systems Dynamics: A Unified Approach,* 2nd edition, John Wiley, NY.

LACERDA E SILVA, A. Q., 2011. *Verificação Formal com Lógica Temporal e Epistêmica*. Monografia, Bacharelado em Ciência da Computação, Universidade de Brasília, Brasília.

MATHWORKS, INC., 2003. *Stateflow and Stateflow Coder User's Guide.*

MATTSSON, S.E.; ELMQVIST, H.; OTTER, M., 1998. *Physical system modeling with Modelica,* Control Engineering Practice, 6 (1998), pp. 501–510

MODELICA ASSOCIATION, 2003. *Modelica- A Unified Object-Oriented Language for Physical Systems Modeling* in Language Specification Version 2.1, 2003.

OGATA, K., 2011. *Engenharia de Controle Moderno*, 5ª Ed., Pearson Education.

OTTER, M.; ARZEN, K. E.; DRESSLER, I.;, 2005. *Stategraph-a modelica library for hierarchical state machines*, in Proc. of the Int. Modelica Conf., pp. 569–578.

OTTER, M.; MALMHEDEN, M.; ELMQVIST, H.; MATTSSON, S. E.; JOHNSSON, C., 2008. *A new formalism for modeling of reactive and hybrid systems*, in Proc. of the Modelica Conf..

PAYNTER, H. M., 1961. *Analysis and Design of Engineering Systems,* MIT Press.

PRADELLA, M., 2009. *A User's Guide to Zot*, in CNR IEIIT.

PRADELLA, M.; MORZENTI, A.; SAN PIETRO, P., 2007. *The symmetry of the past and of the future: bi-infinite time in the verification of temporal properties*, in Proceedings of ESEC/SIGSOFT FSE, pp. 312–320.

SCIAVICCO, L.; SICILIANO, B., 2000. *Robotica industriale – Modellistica e controllo di robot manipolatori*, 2ª ed., Mc Graw-Hill.

ZEIGLER, B. P.; PRAEHOFER, H.; KIM, T. G., 2000. *Theory of Modeling and Simulation*, Second Edition, 2nd ed. Academic Press.

ZEIGLER, B.P.; LEE, J.S, 1998. *Theory of quantized systems: formal basis for DEVS/HLA distributed simulation environment*, SPIE Proceedings, Vol. 3369, pp. 49-58.

ZUPANCIC B.; KARBA R.; ATANASIJEVIC-KUNC M.; MUSIC J., 2008. *Continuous Systems Modelling Education – Causal or Acausal Approach?*, in Information Technology Interfaces, 2008.