Universidade Federal do Amazonas

Faculdade de Tecnologia

Programa de Pós-Graduação em Engenharia Elétrica

Verificação e Síntese de Controladores de Realimentação de Estados Estática com Garantias Formais de Desempenho Não Frágil

Thiago Rodrigo Félix Cavalcante

Manaus – Amazonas

Agosto de 2019

Thiago Rodrigo Félix Cavalcante

Verificação e Síntese de Controladores de Realimentação de Estados Estática com Garantias Formais de Desempenho Não Frágil

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica, como requisito parcial para obtenção do Título de Mestre em Engenharia Elétrica. Área de concentração: Automação e Controle.

Orientador: Prof. D.Sc. Eddie Batista de Lima Filho

Co-orientador: Prof. Ph.D. Lucas Carvalho Cordeiro

Ficha Catalográfica

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).

Cavalcante, Thiago Rodrigo Felix
C376v Verificação e Síntese de Controladores de Realimentação de Estados Estática com Garantias Formais de Desempenho Não Frágil / Thiago Rodrigo Felix Cavalcante. 2019
98 f.: il. color; 31 cm.

Orientador: Eddie Bastista de Lima Filho
Coorientador: Lucas Carvalho Cordeiro
Dissertação (Mestrado em Engenharia Elétrica) - Universidade Federal do Amazonas.

1. Sistemas de Controle. 2. Verificação de Parâmetros de Desempenho. 3. Síntese. 4. Verificação Formal. I. Lima Filho, Eddie Bastista de II. Universidade Federal do Amazonas III. Título

THIAGO RODRIGO FÉLIX CAVALCANTE

VERIFICAÇÃO E SÍNTESE DE CONTROLADORES DE REALIMENTAÇÃO DE ESTADOS ESTÁTICA COM GARANTIAS FORMAIS DE DESEMPENHO NÃO FRÁGIL

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal do Amazonas, como requisito parcial para obtenção do título de Mestre em Engenharia Elétrica na área de concentração Controle e Automação de Sistemas.

Aprovado em 30 de agosto de 2019.

BANCA EXAMINADORA

Prof. Dr. Eddie Batista de Lima Filho, Presidente

R&D/TPV do Brasil

Prof. Dr. Waldir Sabino da Silva Júnior, Membro

Universidade Federal do Amazonas

Prof. Dr. Raimundo da Silva Barreto, Membro

Universidade Federal do Amazonas

À minha família.

Agradecimentos

Primeiramente, agradeço ao Senhor, nosso Deus, pelo dom da vida e por todas as conquistas alcançadas. E principalmente, agradeço Seu amor e misericórdia, pois cuidou e cuida de todas as coisas.

Agradeço à minha noiva, Ingrid Nascimento pelo companheirismo e amor. Sei que você sempre estará comigo, apoiando-me em todos os momentos. Você me inspira a buscar mais as coisas que o Senhor põe em nossos caminhos.

Agradeço à minha mãe, Roseane Félix, ao meu pai, Carlos Cavalcante, e a minha avó, Evanilde da Silva, que me deram todo o suporte, educando-me e sempre me motivando a ir além.

Agradeço aos meus amigos, mais que irmãos que o Senhor colocou em minha vida, na Austrália e espalhados pelo Brasil, em especial a Fernando Gama e Gustavo Carvalho; aos meus amigos da UFAM, Arllem Farias, Breno Linhares, Erickson Alves, Robson Guimarães, Walmir Acioli e Weider Serruia, que foram meus companheiros durante os anos de graduação, mestrado e da vida; aos meus amigos do trabalho, do inTera, em especial Carlos Martins e Sirineo Filho, por todo o apoio.

Agradeço também aos meus orientadores, Prof. Eddie Lima Filho e Prof. Lucas Cordeiro, pela amizade, direcionamento, suporte, paciência e conselhos. Também agradeço ao Prof. Iury Bessa por todo acompanhamento e conselhos acadêmicos e pessoais. Saibam que me espelho em vocês como profissional. Continuem procurando dar um caminho melhor a alunos através da educação.

“Assim, quer vocês comam, quer bebam, quer façam qualquer outra coisa, façam tudo para a glória de Deus.”.

1 Coríntios 10:31

Resumo

Neste trabalho, uma abordagem para a realização de verificação e síntese de sistemas de controle discreto com realimentação de estados foi descrita, levando-se em consideração requisitos de desempenho da resposta ao degrau em sistemas de controle, a qual é baseada em técnicas de síntese indutiva guiada por contraexemplo (*counter-example guided inductive synthesis* - CEGIS). Nesse esquema, requisitos de desempenho (*e.g.*, tempo de assentamento e máximo sobressinal) são avaliados, em um determinado sistema de controle, com o objetivo de verificar se os valores desejados são atendidos. Caso isso não aconteça, torna-se necessário encontrar um sistema que possibilite isso e, nesse caso, um controlador é novamente projetado. Para a geração de controladores, uma técnica de aprendizagem que se baseia em algoritmo genético foi utilizada, onde, a cada iteração em que um requisito não seja satisfeito, sabe-se que o controlador associado não é adequado. Na verificação desses requisitos de desempenho, em sistemas de controle discreto, considerou-se fragilidade (erros de quantização numérica, arredondamentos, efeitos de palavra de máquina finita, etc) nos controladores utilizados. A abordagem desenvolvida é útil para auxiliar engenheiros em seus projetos de sistemas de controle discretos, visto que fragilidades normalmente ocorrem durante implementações em plataformas digitais e, nesse caso, um sistema que atenda os requisitos desejados pode ser gerado. A sua implementação ocorreu dentro da ferramenta DSVerifier, que é baseada em verificação limitada (e ilimitada) de modelos e teorias de módulo de satisfabilidade. A metodologia proposta foi avaliada em um conjunto de padrões de teste clássicos (*benchmarks*) de sistemas de controle, extraídos da literatura, bem como em casos específicos e considerando diferentes autovalores e configurações de controladores. Os resultados experimentais mostram a sua eficácia em síntese de sistemas de controle discreto com realimentação de estados, levando-se em consideração requisitos de desempenho, visto que considera problemas práticos de implementação, o que não

ocorre com outros métodos existentes.

Palavras-chave: Sistemas de Controle, Verificação de Parâmetros de Desempenho, Síntese.

Abstract

In this work, we describe an approach to perform verification and synthesis in discrete control systems with state feedback over step response performance requirements in control systems which is based on counterexample-guided inductive synthesis techniques (CEGIS). In this approach there is a performance requirement (*e.g.*, settling time and maximum overshoot) in a given control system in order to know if it satisfies the desired value for that requirement, if it does not satisfy, one must find a system that satisfies the desired requirement, in which case the system controller is reset. For the generation of the controller, we use a learning technique where each iteration that the verification of the requirement does not satisfy, we learn that this controller is not worthy. In the verification of these performance requirements in discrete control systems, we consider the fragility (numerical quantization error, round-offs, etc.) in the controllers used. This approach is useful for assisting control engineers in their discrete control systems projects since such weaknesses occur during implementation on a digital platform, in which case this approach generates the system that meets the requirements desired in the design. This approach was implemented using DSVerifier which is a tool that employs bounded (and unbounded) model verification based on satisfiability modulo theories. Our approach was evaluated in a set of classical control system benchmarks extracted from the control literature, as well as in specific benchmarks considering different eigenvalues. The experimental results show that the elaborated approach is effective for the synthesis of performance requirements in discrete state feedback control systems since it considers practical implementation problems (FWL effects), unlike other methods that routinely do not consider these problems.

Keywords: Control Systems, Performance Requirement Verification, Synthesis.

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviações

AD - *Analógico Digital*

ATS - *Adaptative Transition System*

BDD - *Banary Decision Diagram*

BIBO - *Bounded Input Bounded Output*

BMC - *Bounded Model Checking*

CEGIS - *Counter Example Guided Inductive Synthesis*

CTL - *Computational Tree Logic*

DA - *Digital Analógico*

FSM - *Finite State Machine*

FWL - *Finite Word Length*

LQR - *Linear Quadratic Regulator*

LTL - *Linear Temporal Logic*

MAP - *Mean Arterial Pressure*

MDD - *Multiple Decision Diagram*

MIMO - *Multiple Input Multiple Output*

PSO - *Particle Swarm Optimization*

PWA - *Piecewise Affine*

QPSO - *Quantic Particle Swarm Optimization*

RAM - *Random Access Memory*

RCLF - *Robust Control LLyapunov-like Function*

SDP - *Semi Definite Programming*

SISO - *Single Input Single Output*

SMT - *Satisfiability Modulo Theories*

Capítulo 1

Introdução

Um sistema de controle digital consiste em sensores, sistema controlado, algoritmos de controle e atuadores, que juntos buscam manter o comportamento das variáveis de uma planta (sistema controlado) sob controle, ou seja, garantem as respostas desejadas de estado estacionário e transitório [1]. O desenvolvimento de controladores digitais é uma tarefa fundamental na engenharia de controle, uma vez que eles são rotineiramente usados para muitas aplicações diferentes, que vão desde aplicações industriais até no ramo aeroespacial [2].

A teoria de controle digital visa preservar algumas propriedades baseadas em modelos de tempo discreto, por exemplo, estabilidade e robustez, que são necessárias para a operação correta de plantas reais através de um controlador digital. Controlar sistemas contínuos usando controladores digitais levanta problemas típicos de sistemas híbridos. Além disso, arredondamento e quantização de amostras e coeficientes em controladores digitais, devido a implementações de comprimento de palavras finitas (*finite word length* - FWL), podem levar a uma condição que ocorre quando um cálculo produz um resultado maior ou menor do que um determinado registrador pode armazenar ou representar (*overflow*) [3], oscilação de ciclo limite e sensibilidade a polos e zeros, o que pode causar instabilidade do sistema e degradação do desempenho [4–6]. De fato, essa sensibilidade é chamada de fragilidade [7] e é importante para o projeto de sistemas de controle, uma vez que está diretamente relacionado à sua robustez. Na literatura, existem alguns estudos relacionados que consideram perturbações nas implementações de sistemas de controle, como os trabalhos de Yordanov *et al.* e Chaves *et al* [8, 9].

Muitas vezes, precisa-se investigar se determinados sistemas de controle atendem a certos requisitos relativos a resposta ao degrau de sistemas de controle. Algumas iniciativas que

utilizam a verificação formal aplicada a sistemas de controle dinâmicos foram desenvolvidas nos últimos anos, como os trabalhos descritos por Bessa *et al.* [6, 10]. Abreu *et al.* [11] também descreveram trabalhos onde aplicaram a verificação formal em sistemas de controle. Outros trabalhos que também utilizaram verificação formal em sistemas de controle foram desenvolvidos [12–15].

Em sistemas de controle, parâmetros de resposta a degraus são comumente usados para especificar um elemento de controle, de modo a indicar que um sistema de controle de malha fechada é seguro em relação às suas especificações de desempenho, ou seja, quando sua resposta ao degrau atende, por exemplo, especificações de tempo assentamento e sobressinal [16]. No entanto, existem alguns estudos relacionados que abordam parâmetros de desempenho com métodos formais como os mostrados por Gross e Kerianne [17] e Jin *et al.* [18].

Também é importante reforçar que os efeitos FWL têm um grande impacto no comportamento do sistema [4]. Geralmente, os controladores são fortemente influenciados por ligeiras imprecisões (por exemplo, erro de arredondamento e quantização), o que, em alguns casos, pode até levar à instabilidade. De fato, quando se implementa um sistema de controle em uma plataforma física (microcontroladores/microprocessadores), inevitavelmente lida-se com efeitos FLW, incluindo quantização de todos os resultados aritméticos (somas e produtos) e sinais de entrada, que afetam notavelmente a localização dos polos e zeros [19].

Como consequência, isso leva a um comportamento diferente, quando comparado com os requisitos de projeto, como se pode perceber ao longo deste trabalho e seus resultados. Além disso, vale notar que o tempo de assentamento e o sobressinal são requisitos típicos de projeto de sistemas de controle, o que leva a um comportamento específico de um sistema em desenvolvimento, juntamente com uma metodologia específica. Eles também são extremamente relacionados uns com os outros, embora sejam geralmente objetivos conflitantes [16].

Geralmente, precisa-se que o sistema se estabilize dentro de um certo período, com um sobressinal específico, e esse requisito revela prontamente a importância de verificar essas propriedades. Nesse sentido, aplicações clínicas geralmente exigem o estabelecimento de especificações de projeto realistas [20] e, por exemplo, sistemas de malha fechada devem responder rápida e suavemente as mudanças nos pontos de referência da pressão arterial média (*mean arterial pressure* - MAP), operação que normalmente é realizado por um anestesista, sem sobressinal excessivo; caso contrário, pode causar sérios problemas em um centro cirúrgico [20].

Nos últimos anos, a literatura trouxe alguns trabalhos que abordam a síntese de progra-

mas, que é uma técnica usada para encontrar entidades que satisfaçam a intenção do usuário expressa em alguma forma de restrições [21]. Uma técnica específica de síntese de programa chamada síntese indutiva guiada por contraexemplo (*Counter-example guided inductive synthesis* - CEGIS) é uma abordagem que determina parâmetros desconhecidos dentro de programas parciais, de forma que os elementos resultantes satisfaçam algumas propriedades de correção [22]. Alguns trabalhos foram feitos utilizando a técnica CEGIS para a realização de síntese, como a descrita por Abate *et al.* [23], onde mostram os pontos fortes de um sintetizador indutivo guiado por contraexemplos comparado com a abordagem de um solucionador de teoria, explorando o espaço da solução de maneira mais eficiente sem depender da orientação do usuário.

Trabalhos que aplicam CEGIS para estabilizar controladores podem ser encontrados na literatura, como o descrito por Ravanbakhsh *et al.* [22] onde é investigado o problema de sintetizar controladores de comutação para estabilizar a planta de tempo contínuo. No entanto, em sistemas de controle, é importante cuidar dos requisitos de desempenho, como tempo de assentamento e sobressinal, de modo a satisfazer um comportamento específico [24].

Problemas relativos a efeitos FWL na implementação de controladores digitais em plataformas do mundo real devem ser levados em consideração quando se projeta controladores digitais, conforme falado anteriormente nos trabalhos [5, 6]. Portanto, deve-se considerar esses fatores quando se sintetiza controladores digitais para que o funcionamento do sistema ocorra de acordo com o esperado.

Visto que a verificação e síntese de controladores digitais apresenta uma lacuna referente a aspectos de desempenho a resposta ao degrau e de implementação (efeitos FWL), este trabalho visa resolver o problema levantado nos últimos parágrafos e apresenta uma metodologia formal para verificar e consequentemente sintetizar um controlador, avaliando os requisitos de desempenho em sistemas de controle digital tais como tempo de assentamento e sobressinal, utilizando uma técnica baseada em CEGIS e algoritmos genéticos [25], considerando os efeitos FWL, que foi implementado e codificado em um verificador de modelo limitado chamado DSVerifier [26, 27].

Algoritmos genéticos são uma classe particular de algoritmos evolutivos que usam técnicas inspiradas pela biologia evolutiva como hereditariedade, mutação, seleção natural e recombinação (ou *crossing over*) [25]. Neste trabalho utiliza-se este algoritmos com o objetivo de gerar um controlador candidato que satisfaça aos requisitos de tempo de assentamento e sobressinal, sendo uma das etapas na metodologia de síntese desenvolvida. A metodologia de síntese

desenvolvida desta pesquisa visa sintetizar um controlador que também satisfaça aos requisitos oriundos dos efeitos FWL.

Assim, a metodologia proposta utiliza técnicas de síntese de programas, algoritmos genéticos e verificador formal para verificar a validade do controlador sintetizado.

1.1 Objetivos

O principal objetivo desta dissertação é propor uma metodologia para sintetizar controladores digitais obedecendo requisitos de desempenho e levando em conta a ação dos efeitos FWL nos controladores.

Os objetivos específicos são:

- Propor uma abordagem para a verificação de tempo de assentamento da resposta ao degrau através da elaboração de algoritmos e de cálculo de invariantes para avaliação dessa propriedade;
- Propor uma abordagem para a verificação de sobressinal da resposta ao degrau através de algoritmos pensados para esse propósito e realizar a avaliação dessa propriedade;
- Integrar os resultados acima em um esquema unificado, que seja capaz de gerar controladores já adequados aos requisitos de sobressinal e tempo de assentamento.

1.2 Contribuições

Dentre as principais contribuições apresentadas pela presente dissertação, pode-se destacar as seguintes:

Durante a fase de concepção deste trabalho, foi verificado uma lacuna em trabalhos disponíveis na literatura, como será mostrado em capítulos posteriores. Essa lacuna é referente a análise de sistemas de controle com respeito a aspectos de desempenho. A primeira contribuição foi o desenvolvimento de uma metodologia cujo objetivo é realizar a verificação de sistemas de controle que atendam a requisitos de tempo de assentamento. A segunda contribuição refere-se a metodologia de verificação de sobressinal que por sua vez, também foram encontrados poucos trabalhos que tratam desse requisito funcional em sistemas de controle. Após isso, foi

pensado na possibilidade de integrar essas *engines* de verificação em um esquema onde pudesse gerar um controlador capaz de fazer o sistema de controle atender requisitos de desempenho. Portanto, a terceira contribuição foi a criação de um mecanismo de síntese de controladores, onde foi integrado a metodologia formal de verificação de tempo de assentamento como uma das etapas do processo de síntese. A quarta contribuição foi a integração, também, da *engine* de verificação do máximo sobressinal em sistemas de controle, sendo um módulo integrado a um esquema de síntese que gera um controlador que atende a essa propriedade. Também foi desenvolvido um esquema que dá suporte a síntese de controladores que atendam a requisitos de tempo de assentamento e sobressinal ao mesmo tempo. Uma das etapas do processo de síntese, desenvolvido aqui, é a geração de controladores candidatos onde utilizou-se algoritmo genético para este fim. As metodologias de síntese desenvolvidas neste trabalho foram baseadas na técnica de CEGIS. Por último, foi criado uma base de dados diversificada com um conjunto de *benchmarks* capaz de validar e testar as metodologias desenvolvidas nesta dissertação.

1.3 Organização da Dissertação

A dissertação está organizada da seguinte maneira: no Capítulo 2 são apresentados os conceitos básicos de verificação formal, sistemas dinâmicos lineares, técnicas de projeto de controle para sistemas discreto e, ao fim, fala-se a respeito de implementação de controladores digitais no mundo real; o Capítulo 3 apresenta um resumo dos trabalhos relacionados à verificação formal aplicada a controladores de sistemas de controle, à síntese de controladores de sistemas de controle e no fim do capítulo, os trabalhos são comparados, salientando as principais diferenças entre a abordagem proposta neste trabalho em relação às existentes; o Capítulo 4 descreve o método proposto para sintetizar controladores digitais, considerando aspectos de implementação, como os efeitos FWL; no Capítulo 5 são apresentados os resultados experimentais feitos em um conjunto de *benchmarks* e também é feita uma discussão dos resultados obtidos; e por fim o Capítulo 6 apresenta as conclusões do trabalho, além de apresentar sugestões para trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Neste capítulo, são apresentados os conceitos básicos utilizados durante o desenvolvimento desta dissertação. Primeiramente, serão mostrados alguns conceitos importantes sobre verificação formal com o objetivo de se mostrar o conceito de invariantes. Em seguida, apresentam-se representações de sistemas dinâmicos lineares (contínuos e discretos), processo de discretização, e análise quantitativa da resposta ao degrau de sistemas dinâmicos. Também será mostrado técnicas de projeto de controle para sistemas discretos, em particular, alocação de polos, regulador quadrático linear (*linear quadratic regulator* - LQR) e CEGIS. Além disso, uma breve introdução a implementação de controladores digitais e efeitos de palavra finita. Por fim, um resumo do capítulo é dado, para sintetizar o conteúdo do mesmo.

2.1 Verificação Formal

A verificação formal é o processo de verificar se um projeto satisfaz alguns requisitos ou até mesmo propriedades [28]. Normalmente, preocupa-se com a verificação formal de projetos que podem ser especificados hierarquicamente. Isso também é consistente com a forma como um ser humano faria para projetar. Comumente, para verificar formalmente um projeto, ele deve primeiro ser convertido em um formato "verificável" mais simples. O projeto é especificado como um conjunto de sistemas de interação. Cada um tem um número finito de configurações, chamados estados. Estados e transição entre estados constituem máquinas de estado finito (*finite state machines* - FSMs) [29]. O primeiro passo na verificação consiste em obter uma descrição completa do sistema FSM. Dado um estado atual (ou configuração atual), o próximo estado (ou

configuração sucessiva) de uma FSM pode ser escrito como uma função de seu estado atual e entradas (função de transição ou relação de transição).

Nota-se que todo este processo falado anteriormente comporta-se como funções discretas. Funções discretas podem ser representadas convenientemente por diagramas de decisão binária (*binary decision diagrams* - BDDs) - uma estrutura de dados que representa funções booleanas (de 2 valores), e seus diagrama de decisão de múltiplos valores de extensão (*multiple decision diagrams* - MDDs), uma estrutura de dados que representa funções discretas de valor finito [30]. Usa-se BDDs e MDDs para representar todas as quantidades necessárias neste espaço discreto (mais especificamente as funções de transição, as entradas, as saídas e os estados dos FSMs). Para que os BDDs e MDDs sejam representações eficientes de funções discretas, uma boa ordenação de variáveis de entrada (entradas reais, saídas e estado) das funções deve ser calculada. Em geral, os BDDs operam em conjuntos de pontos em vez de pontos individuais que por sua vez é chamado de manipulação simbólica.

Os dois métodos mais populares para verificação formal automática são a contenção de linguagem [31] e a verificação de modelo [28]. Muitos problemas na verificação formal podem ser formulados em uma estrutura de contenção de linguagem. Decidir se a linguagem de um autômato M está contida na de P, ou seja, $L(M) \subseteq L(P)$, é PSPACE completo [32] se P é não determinístico. A contenção da linguagem é geralmente testada pela conversão a uma pergunta de *emptiness language* [33]: um autômato $\bar{P}$ aceitando a linguagem $L(\bar{P})$ é construído e é determinado se $L(M \times \bar{P})$ está vazio. As formas assintoticamente ótimas conhecidas de obter tal $\bar{P}$ requerem a determinação de P primeiro [34]. Verificação de modelos é uma técnica de verificação automática para sistemas concorrentes de estados finitos [35].

Neste trabalho, utiliza-se uma ferramenta (DSVerifier) que utiliza a técnica de verificação de modelos limitada (*bounded model cheching* - BMC) que é uma técnica que utiliza um solucionador SAT (*satisfiability* - satisfabilidade) proposicional em vez de técnicas de manipulação de BDD [36]. Desde a sua introdução em 1999, o BMC tem sido bem recebido pela indústria. Ele pode encontrar muitos erros lógicos em sistemas complexos que não podem ser encontrados por técnicas concorrentes e, portanto, é amplamente percebido como uma técnica complementar à verificação de modelos baseada em BDD. A ideia básica no BMC é procurar um contraexemplo em execuções cujo comprimento é limitado por algum inteiro k. Se nenhum *bug* for encontrado, então incrementa k até que um *bug* seja encontrado, ou o problema se torne intratável ou algum limite superior pré-conhecido seja alcançado (este limite é chamado de

limite de completude do projeto).

A geração de invariantes é uma técnica muito útil que ajuda a indução a fortalecer a propriedade, mas às vezes pode ter alto custo de memória e tempo. Alguns verificadores de modelo verificam todas as propriedades ao mesmo tempo (verificação incremental), para que a prova de uma propriedade "fácil" possa ajudar na prova de uma difícil. Nesse sentido, será introduzido, na próxima seção, o conceito mais detalhado de invariantes.

2.1.1 Invariantes

Nos últimos anos, invariantes desempenham um papel importante na engenharia de *software*, como teste e verificação de *software* e sistemas [37–39]. Invariantes são propriedades de variáveis de programa e relacionamentos entre essas variáveis em uma linha específica de código que se chama de ponto de programa. A geração de invariantes é uma chave significativa na verificação de um programa. Essas propriedades e relacionamentos entre as variáveis ou constantes do programa são sempre verdadeiras; assim, programador ou testador pode estimar o comportamento de um programa em diferentes pontos do programa. Invariante também é usado na geração de modelo comportamental de *software* [40].

Gadelha *et al.* [41] implementam uma ferramenta de verificação que usa invariantes baseado em intervalos que são introduzidos automaticamente no programa como suposições e, embora a implementação tenha algumas limitações em manter o controle das relações entre variáveis, reforça significativamente os resultados do algoritmo k-*induction*. Os autores observaram que o uso de invariantes aumenta o número de provas corretas em cerca de 7% em relação aos *benchmarks* da competição em verificação de *software* (*Competition on Software Verification* - SV-COMP) [42]. Há outros trabalhos que utilizam o conceito de invariantes com este mesmo objetivo, como o descrito por Rocha *et al.* [43]. Na presente dissertação, utiliza-se uma invariante com o objetivo de ajudar na verificação do tempo de assentamento de sistemas de controle digital, como será demonstrado nos capítulos posteriores.

2.1.2 DSVerifier

A presente dissertação implementa sua metodologia na ferramenta DSVerifier [9,26] que é uma ferramenta de verificação de sistemas digitais que usa o verificador de modelos limitado ESBMC [44] como a *engine* de verificação principal. recebe uma especificação do sistema de

controle digital, calcula os números representáveis máximo e mínimo para um formato de FWL escolhido e, em seguida, durante a verificação, verifica se todos os parâmetros necessários foram corretamente fornecidos. No final, o DSVerifier adiciona chamadas explícitas a sua *engine* de verificação (*e.g.*, `__ESBMC_assume` and `__ESBMC_assert`), a fim de verificar violações de propriedades [44].

DSVerifier usa a técnica de verificação BMC baseada na teoria de modulo satisfatibilidade (*satisfiability modulo theories* - SMT) e também implementa uma regra de prova eficiente *k*-induction [45–47]. Até então, suporta estabilidade, ciclo limite e erro de quantização, para sistemas de malha fechada que sofrem efeitos de FWL e, além disso, *overflow* e fase mínima, para aqueles de malha aberta [26].

A Figura 2.1 mostra um mecanismo de verificação e, consequentemente, síntese, onde os parâmetros de entrada são as especificações do sistema em análise e especificações de implementação (efeitos de FWL). Essas especificações são repassadas parada a ferramenta DSVerifier que, por sua vez trabalha em conjunto com a ferramenta de verificação ESBMC.

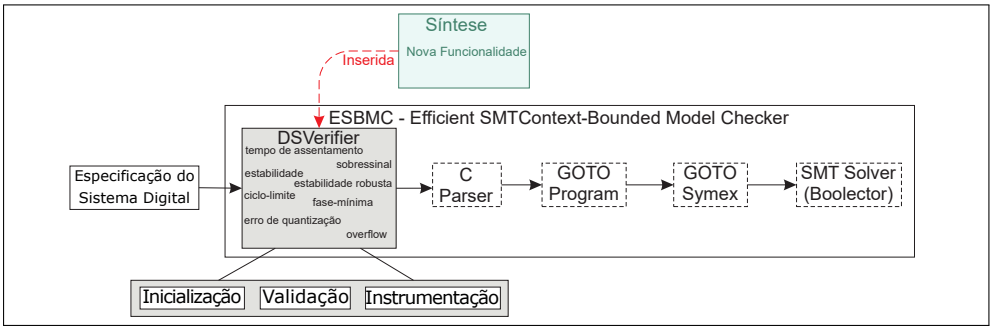


Figura 2.1: Suporte a síntese no DSVerifier.

A metodologia proposta neste trabalho para verificar e sintetizar controladores digitais que atendam a requisitos de tempo de assentamento e sobressinal foi implementada no DSVerifier. Neste caso, foi inserido no DSVerifier o suporte a verificação das propriedades de tempo e assentamento e sobressinal, bem como a realização de síntese de controladores tendo essas propriedades como requisito.

2.2 Sistemas Dinâmicos Lineares

Sistemas dinâmicos lineares são sistemas dinâmicos [48] cujas funções de avaliação são lineares [49]. Embora os sistemas dinâmicos, em geral, não tenham soluções de forma fechada,

os sistemas dinâmicos lineares podem ser resolvidos com exatidão e possuem um rico conjunto de propriedades matemáticas. Sistemas lineares também podem ser usados para entender o comportamento qualitativo de sistemas dinâmicos gerais, calculando os pontos de equilíbrio do sistema e aproximando-o como um sistema linear em torno de cada um desses pontos [48].

Em um sistema dinâmico linear, a variação de um vetor de estado (um vetor N-dimensional dado por x) é igual a uma matriz constante (dado por $\mathbf{A}$) multiplicada por $\mathbf{x}$. Essa variação pode ter duas formas: ou como um fluxo, no qual $\mathbf{x}$ varia continuamente com o tempo [50]

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) \tag{2.1}$$

ou como um mapeamento, em que $\mathbf{x}$ varia em etapas discretas [50]

$$\mathbf{x}_{n+1} = \mathbf{A}\mathbf{x}_n \tag{2.2}$$

Estas equações são lineares no seguinte sentido: se $\mathbf{x}(t)$ e $\mathbf{y}(t)$ são duas soluções válidas, então também é qualquer combinação linear [51] das duas soluções, por exemplo:

$$\mathbf{z}(t) \stackrel{\text{def}}{=} \alpha\mathbf{x}(t) + \beta\mathbf{y}(t),$$

onde α e β são quaisquer dois escalares. A matriz $\mathbf{A}$ não precisa ser simétrica [52].

Sistemas dinâmicos lineares podem ser resolvidos exatamente, em contraste com a maioria dos não-lineares. Ocasionalmente, um sistema não linear pode ser resolvido exatamente por uma mudança de variáveis para um sistema linear. Além disso, as soluções de (quase) qualquer sistema não linear podem ser bem aproximadas por um sistema linear equivalente próximo de seus pontos fixos [53]. Assim, entender os sistemas lineares e suas soluções é um primeiro passo crucial para entender os sistemas não-lineares mais complexos.

2.2.1 Contínuos

Um sistema é chamado de sistema em tempo contínuo se tiver sinais de tempo contínuo como entrada e gerar sinais de tempo contínuo como saída [54]. A entrada será indicada por itálico em minúsculo $u(t)$ para entrada única ou por negrito $\mathbf{u}(t)$ para múltiplas entradas. Se o sistema tem p entradas, então $\mathbf{u}(t)$ é um vetor $p \times 1$ ou $\mathbf{u} = [u_1\ u_2\ \ldots\ u_p]^T$, onde T denota a transposição. Da mesma forma, a saída será denotada por $y(t)$ ou $\mathbf{y}(t)$. Supõe-se que o tempo t varie de $-\infty$ a ∞.

Equações de Espaço de Estados

Todo sistema linear invariante no tempo pode ser descrito por:

$$\begin{cases} \dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) \\ \mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t) \end{cases}, \tag{2.3}$$

Para um sistema com p entradas, q saídas e n variáveis de estado, $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ e $\mathbf{D}$ são, respectivamente, $n \times n$, $n \times p$, $q \times n$ e $q \times p$ matrizes constantes.

Um sistema de tempo contínuo ou discreto é linear se as propriedades de aditividade e homogeneidade se mantiverem [55]. A resposta de um sistema linear pode ser decomposta como:

$$y = y_{\mathrm{x0}} \ + \ y_{\mathrm{u0}}$$

onde y é a resposta geral, y_{x0} é a resposta de estado nulo e y_{u0} é a resposta de entrada nula. E as respostas de estado nulo satisfazem a propriedade de superposição, e portanto, as respostas de entrada nula também [55].

2.2.2 Discretos

A maioria dos conceitos em sistemas de tempo contínuo pode ser aplicada diretamente aos sistemas de tempo discreto. Um sistema é chamado de sistema de tempo discreto se aceitar sinais de tempo discreto como entrada e gerar sinais de tempo discreto como saída [55]. Todos os sinais de tempo discreto em um sistema serão considerados como tendo o mesmo período de amostragem T_{s}. A entrada e a saída serão denotadas por $u[k] = u(kT)$ e $y[k] = y(kT)$, onde k denota o instante de tempo discreto e é um inteiro que varia de $-\infty$ a ∞. Eles se tornam negrito para múltiplas entradas e múltiplas saídas [56].

Um sistema de tempo discreto é causal se a saída atual depende de entradas atuais e passadas [57]. O estado no tempo k_0, denotado por $\mathbf{x}[k0]$, é a informação no instante de tempo k_0, que com $\mathbf{u}[k]$, $k \geq k_0$, determina unicamente a saída $\mathbf{y}[k]$, $k \geq k0$. As entradas de $\mathbf{x}$ são chamadas de variáveis de estado. Se o número de variáveis de estado for finito, o sistema de tempo discreto é agrupado; caso contrário, é distribuído. Todo sistema de tempo contínuo envolvendo atraso de tempo é um sistema distribuído. Em um sistema de tempo discreto, se o

atraso de tempo for um múltiplo inteiro do período de amostragem T_s, então o sistema de tempo discreto é um sistema aglomerado [56].

Equações de Espaço de Estados

Cada sistema de tempo discreto linear invariante no tempo pode ser descrito por:

$$\Omega : \begin{cases} \mathbf{x}[k+1] = \mathbf{A}\mathbf{x}[k] + \mathbf{B}\mathbf{u}[k] \\ \mathbf{y}[k] = \mathbf{C}\mathbf{x}[k] + \mathbf{D}\mathbf{u}[k] \\ \mathbf{u}[k] = \mathbf{r}[k] - \mathbf{K}\mathbf{x}[k] \end{cases}, \tag{2.4}$$

onde **A**, **B**, **C**, **D** e **K** são matrizes contantes e **r** é a referência de entrada. Seja $\hat{\mathbf{x}}(z)$ a transformada Z de $\mathbf{x}[k]$ ou

$$\hat{\mathbf{x}}(z) = Z[\hat{\mathbf{x}}[k]] = \sum_{k=0}^{\infty} \mathbf{x}[k] z^{-k}$$

Tem-se que

$$\begin{aligned} Z[\hat{\mathbf{x}}[k+1]] &= \sum_{k=0}^{\infty} \mathbf{x}[k+1] z^{-k} \\ &= z \left[\sum_{l=1}^{\infty} \mathbf{x}[l] z^{-l} + \mathbf{x}[0] - \mathbf{x}[0] \right] = z\left(\hat{\mathbf{x}}(z) - \mathbf{x}[0]\right) \end{aligned}$$

Aplicando a transformada Z na equação (2.4) e rearranjando, tem-se

$$\hat{\mathbf{x}}(z) = (z\mathbf{I} - \mathbf{A})^{-1} z\mathbf{x}[0] + (z\mathbf{I} - \mathbf{A})^{-1} \mathbf{B}\hat{\mathbf{u}}(z)$$

$$\hat{\mathbf{y}}(z) = \mathbf{C}(z\mathbf{I} - \mathbf{A})^{-1} z\mathbf{x}[0] + \mathbf{C}(z\mathbf{I} - \mathbf{A})^{-1} \mathbf{B}\hat{\mathbf{u}}(z) + \mathbf{D}\hat{\mathbf{u}}(z)$$

A análise em espaço de estados é um excelente método para o projeto e análise de sistemas de controle. O método convencional e antigo para o projeto e análise de sistemas de controle é o método da função de transferência. O método da função de transferência para projeto e análise tem muitas desvantagens. A função de transferência é definida sob condições iniciais nulas. A abordagem da função de transferência pode ser aplicada apenas a sistemas lineares invariantes no tempo. Também não dá nenhuma ideia sobre o estado interno do sistema, bem como não pode ser aplicado a sistemas com múltiplas entradas e múltiplas saídas (*multiple-input multiple-output* - MIMO) [57]. É comparativamente difícil executar a análise da função de transferência nos computadores. E devido a esses fatores, principalmente o fato de poder

trabalhar com sistemas MIMO e ser mais acessível sua implementação em computadores, o desenvolvimento desse trabalho será feito utilizando a análise em espaço de estados.

2.2.3 Estabilidade de Sistemas Lineares Invariantes no Tempo

Os sistemas são projetados para executar algumas tarefas ou para processar sinais. Se um sistema não é estável, o sistema pode ocasionar até mesmo desastres (queimar, desintegrar ou saturar quando um sinal, não importa quão pequeno seja, for aplicado). Os sinais de saída dos sistemas devem ser limitadas, ou seja, sinais não periódicos e concentrados em intervalos de tempo com duração bem definida. Portanto, na prática, um sistema instável é inútil e a estabilidade é um requisito básico para todos os sistemas. Além da estabilidade, os sistemas devem atender a outros requisitos, como rastrear os sinais desejados e suprimir o ruído, para serem realmente úteis, na prática. A resposta de sistemas lineares sempre pode ser decomposta como a resposta de estado nulo e a resposta de entrada nula. É comum na literatura mostrar a estabilidade destas duas respostas separadamente. Nesta seção é feita uma breve introdução de estabilidade com entrada limitada e saída limitada (*bounded-input bounded-output* - BIBO) para a resposta do estado nulo e estabilidades marginais e assintóticas para a resposta da entrada nula, para o caso de sistemas em tempo discreto [55].

Dado o sistema em tempo discreto com uma entrada e uma saída (*single-input single-output* - SISO) [57],

$$\mathbf{y}[k] = \sum_{m=0}^{k} g[k-m]\mathbf{m}[m] = \sum_{m=0}^{k} g[m]\mathbf{u}[k-m], \tag{2.5}$$

onde $g[k]$ é a sequência de resposta ao impulso ou a sequência de saída excitada por uma sequência de impulso aplicada em $k = 0$. Para ser descrito pela equação (2.5), o sistema de tempo discreto deve ser linear, invariante no tempo, e causal. Além disso, o sistema deve ser inicialmente relaxado em $k = 0$.

Diz-se que uma sequência de entrada $\mathbf{u}[k]$ é limitada se $\mathbf{u}[k]$ não crescer até infinito positivo ou negativo, ou se existir uma constante $\mathbf{u}[m]$ tal que $|\mathbf{u}[k]| \leq \mathbf{u}[m] < \infty$, para $0 \leq k < \infty$.

Diz-se que um sistema é BIBO estável se toda sequência de entrada limitada excita uma sequência de saída limitada [24]. Essa estabilidade é definida para a resposta do estado nulo e é aplicável somente se o sistema estiver inicialmente relaxado.

O sistema em tempo discreto mostrado na equação (2.2) é dito ser marginalmente estável ou estável no sentido de Lyapunov se todo estado inicial finito $\mathbf{x}_o$ excita uma resposta limitada [58]. É assintoticamente estável se todo estado inicial finito excita uma resposta limitada, que, além disso, se aproxima de 0 como $k \to \infty$ [58].

Outra forma bastante comum de verificar a estabilidade de um sistema de tempo discreto é através da avaliação dos seus autovalores, ou seja, caso a magnitude dos autovalores da matriz de estado $\mathbf{A}$ for menor que 1, o sistema é dito estável [58], o que pode ser verificado por:

$$\det(\mathbf{A}\lambda - I) = 0, \tag{2.6}$$

onde resolvendo a equação, consegue-se encontrar os autovalores de $\mathbf{A}$.

Se fizer uma análise do sistema resultante no domínio da frequência, os autovalores em malha fechada correspondem aos polos da função de transferência em malha fechada.

2.2.4 Processo de Discretização

Segundo Smith *et al.* [59], a maioria dos sinais encontrados diretamente na ciência e na engenharia são contínuo: intensidade da luz que muda com a distância, tensão que varia ao longo do tempo, uma taxa de reação química que depende da temperatura, etc. A Conversão analógico-digital (*analog-to-digital conversion* - ADC) e a conversão digital-analógica (*digital-to-analog conversion* - DAC) são os processos que permitem que os computadores digitais interajam com esses sinais cotidianos.

O ADC é um dispositivo físico que converte uma amplitude de tensão ou corrente em sua entrada em um código binário representando um valor de amplitude quantizado mais próximo da amplitude da entrada. Sob o controle de um *clock* externo, o ADC pode ser iniciado e concluir uma conversão analógico-digital a cada T segundos. No entanto, a conversão não é instantânea e, por essa razão, um sistema analógico-digital de alto desempenho normalmente inclui a amostragem e retenção, como, por exemplo, retenção de ordem zero (*zero-order-hold* - ZOH). A saída do ZOH é uma forma de onda de escada, onde os valores da amostra são mantidos constantes durante o período de amostragem de T segundos. O quantizador é um sistema não-linear cuja finalidade é transformar a amostra de entrada $x[n]$ em um conjunto finito de valores prescritos (a operação é representada como $\hat{x}[n] = Q(x[n])$, onde $\hat{x}[n]$ é a amostra quantizada). Um exemplo de quantizador é o arredondamento para o nível de quantização

mais próximo. No codificador, geralmente, os 2^{B+1} níveis podem ser codificados com um código binário de $(B+1)$ *bits*. Em princípio, qualquer atribuição de símbolos pode ser usada, e muitos esquemas binários de codificação existem, cada um com suas próprias vantagens e desvantagens, dependendo da aplicação.

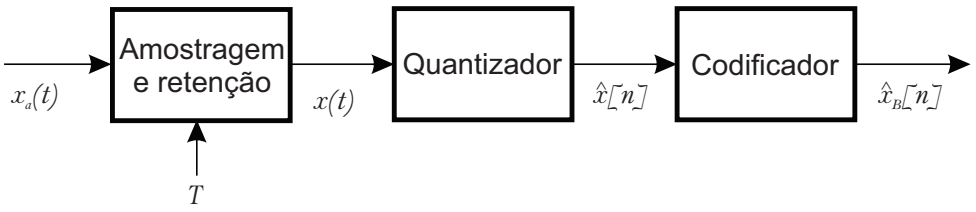


Figura 2.2: Diagrama em blocos do funcionamento de um ADC.

Para fazer o processo inverso, ou seja, a conversão digital-analógica, como mostrado na Figura 2.3, um DAC pega uma sequência de palavras de código binário como sua entrada e produz uma saída de tempo contínuo da forma $x_{\mathrm{DA}}(t) = \sum_{n=-\infty}^{\infty} \hat{x}[n] h0(t-nT)$, onde $h_0(t)$ é a resposta ao impulso do ZOH. O DAC segura a amostra quantizada para um período de amostragem da mesma maneira que o ZOH mantém a amostra de entrada não quantificada. Depois, tem-se uma sequência conceitual de impulsos que são processados por um filtro de reconstrução usando alguma forma de interpolação para preencher os dados entre os impulsos. Um DAC prático convencional converte os números em uma função constante por partes compostas de uma sequência de funções retangulares que é modelada com o ZOH. Outros métodos de digital-analógico, baseados na modulação delta-sigma, produzem uma saída modulada de densidade de pulso que pode ser filtrada de forma semelhante para produzir um sinal que varia suavemente.

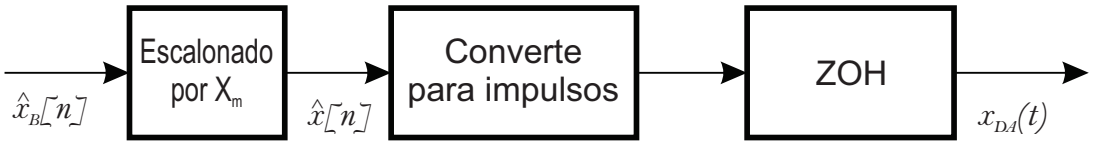


Figura 2.3: Diagrama em blocos do funcionamento de um DAC.

De acordo com o teorema de amostragem de Nyquist-Shannon, um DAC pode reconstruir o sinal original a partir dos dados amostrados desde que sua largura de banda atenda a certos requisitos (por exemplo, um sinal de banda base com largura de banda inferior à frequência de Nyquist ou em se tratando da teoria da amostragem de sinais passa-banda, onde se leva em consideração a largura de bando do sinal). A amostragem digital introduz erros de quantização que se manifestam como ruído de baixo nível no sinal reconstruído.

Considerando o sistema descrito na equação (2.3), se o conjunto de equações é para ser computado em um computador digital, ele deve ser discretizado.
Dado que $\dot{\mathbf{x}}(t) = \lim_{T_s \to 0} \frac{\mathbf{x}(t+T_s)-\mathbf{x}(t)}{T_s}$, pode-se aproximar equação (2.3), usando $t = kT_s$ (aproximação de Euler), para $k = 0, 1, \ldots$, de

$$\begin{cases} \dot{\mathbf{x}}((k+1)T_s) = (\mathbf{I} + T_s\mathbf{A})\mathbf{x}(kT_s) + T_s\mathbf{B}\mathbf{u}(kT_s) \\ \mathbf{y}(kT_s) = \mathbf{C}\mathbf{x}(kT_s) + \mathbf{D}\mathbf{u}(kT_s) \end{cases}.$$

Esta é uma equação de espaço de estados em tempo discreto e pode ser facilmente computada em um computador digital. Esta discretização é a mais fácil de realizar, mas produz os resultados menos precisos para o mesmo T_s. A seguir mostra-se uma discretização diferente.

Se uma entrada $\mathbf{u}(t)$ é gerada por um computador digital seguido por um conversor de digital para analógico (ADC), então $\mathbf{u}(t)$ será constante por partes. Essa situação geralmente surge no controle por computador dos sistemas de controle. Seja

$$\mathbf{u}(t) = \mathbf{u}(kT_s) =: \mathbf{u}[k], \ \textit{para } kT_s \leq t < (k+1)T_s \tag{2.7}$$

De acordo com Chen [58], pode-se computar $\mathbf{x}[k+1]$, como

$$\mathbf{x}[k+1] = e^{\mathbf{A}T_s}\mathbf{x}[k] + \left(\int_0^{T_s} e^{\mathbf{A}\alpha} d\alpha\right)\mathbf{B}\mathbf{u}[k]$$

Assim, se uma entrada altera o valor apenas em instantes de tempo discreto kT_s e se computar apenas as respostas em $t = kT_s$, então a equação (2.3) se torna:

$$\begin{cases} \mathbf{x}[k+1] = \mathbf{A}_d\mathbf{x}[k] + \mathbf{B}_d\mathbf{u}[k] \\ \mathbf{y}[k] = \mathbf{C}_d\mathbf{x}[k] + \mathbf{D}_d\mathbf{u}[k] \end{cases}, \tag{2.8}$$

onde $\mathbf{A}_d =$, $\mathbf{B}_d = \int_0^{T_s} e^{\mathbf{A}\tau} d\tau \mathbf{B}$, $\mathbf{C}_d = \mathbf{C}$ e $\mathbf{D}_d = \mathbf{D}$

Segundo Chen [58], a matriz $\mathbf{B}_d$ pode ser obtida por $\mathbf{B}_d = \mathbf{A}^{-1}(\mathbf{A}_d - \mathbf{I})\mathbf{B}$ (caso $\mathbf{A}$ não seja singular).

A solução das equações de estado para sistemas em tempo discreto pode ser obtida por:

$$\mathbf{x}[k] = \mathbf{A}^k\mathbf{x}[0] + \sum_{m=0}^{k-1} (\mathbf{A}^{k-m-1}\mathbf{B}\mathbf{u}[m]) \tag{2.9}$$

$$\mathbf{y}(k) = \mathbf{C}\mathbf{A}^k\mathbf{x}[0] + \sum_{m=0}^{k-1}(\mathbf{C}\mathbf{A}^{k-m-1}\mathbf{B}\mathbf{u}[m]) + \mathbf{D}\mathbf{u}[k] \tag{2.10}$$

Portanto, essas equações são úteis quando se trabalha com a implementação de sistemas em tempo discreto, pois as quantidades calculadas por elas são utilizadas para processar informações que geram resultados de interesse.

2.2.5 Análise Quantitativa da Resposta ao Degrau

Dado um sistema linear de entrada/saída, a forma geral da solução da equação (2.4) é dada pela equação de convolução na equação (2.10) [56]. A partir da forma dessa equação, pode ser visto que a solução consiste em uma resposta de condição inicial e uma resposta de entrada. A resposta de entrada, corresponde aos dois últimos termos da equação (2.10), consiste em dois componentes, a resposta transitória e a resposta em estado estacionário. A resposta transitória ocorre no primeiro período após a entrada ser aplicada e reflete a incompatibilidade entre a condição inicial e a solução de estado estacionário. A resposta de estado estacionário é a parte da resposta de saída que reflete o comportamento de longo prazo do sistema sob as entradas dadas. Para entradas que são periódicas, a resposta em estado estacionário será frequentemente periódica e, para entradas constantes, a resposta será frequentemente constante.

Uma forma de entrada particularmente comum é o degrau unitário, que representa uma mudança abrupta na entrada de um valor para outro. Um degrau unitário (às vezes chamada de função de degrau Heaviside [55]) é definida como:

$$\mathbf{u}[k] = \begin{cases} 0 \; k < 0 \\ \\ 1 \; k \geq 0 \end{cases}$$

Pode-se computar a resposta ao degrau de um sistema linear discreto através da equação (2.10). Usando algumas operações algébricas, pode-se chegar a conclusão de que a resposta em estado estacionário ao degrau unitário é dado por [16]

$$y_{\text{ss}} = \mathbf{C}(\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} + \mathbf{D}, \tag{2.11}$$

se $\mathbf{A}$ possui autovalores com valor de magnitude menor ou igual a 1 (implicando que a origem é um ponto de equilíbrio estável na ausência de qualquer entrada).

A Figura 2.4 ilustra a resposta ao degrau de um sistema discreto. Vários termos são usados ao se referir a uma resposta ao degrau. O valor do estado estacionário y_{ss} de uma resposta ao degrau é o nível final da saída, supondo que converge [24]. O sobressinal M_{p} é a porcentagem do valor final pelo qual o sinal inicialmente se eleva acima do valor final [24]. Isso geralmente pressupõe que os valores futuros do sinal não excedem o valor final em mais do que esse transitório inicial, caso contrário, o termo pode ser ambíguo. Para se chegar ao valor de M_{p}, é necessário saber y_{p} que é o valor do maior pico da resposta do sistema, onde esse valor tem o mesmo sinal ($+$ ou $-$) de y_{ss}; e k_{p} é a amostra onde esse valor de pico está alocado. O tempo de assentamento k_{s} é o tempo necessário para o sinal ficar dentro de uma faixa de valores compreendidos $p\%$ acima e abaixo do valor final (neste trabalho tratado como região de assentamento Π) para todos os tempos futuros. O tempo de assentamento também é por vezes definido como atingindo 1% ou 5% do valor final. Em geral, essas medidas de desempenho podem depender da amplitude do degrau de entrada, mas, para sistemas lineares, algumas quantidades definidas acima (M_{p} e k_{s}) são independentes do tamanho do degrau. Por último, para este trabalho, definidos também o que chamados de tempo de alcance k_{r} que é a amostra onde a resposta do sistema alcança pela primeira vez a região de assentamento.

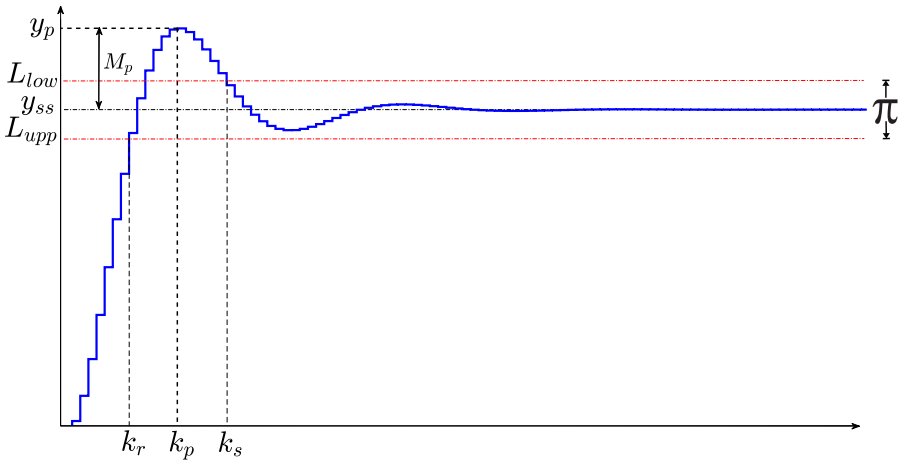


Figura 2.4: Resposta ao degrau de um sistema em tempo discreto.

Caso os requisitos de tempo de assentamento e sobressinal não sejam atendidos, ocorrerá uma violação de especificações funcionais. Por exemplo, em sistemas de telecirurgias que utilizam robôs que necessitam de alta precisão para realizar cirurgias, onde caso essas especificações funcionais sejam violadas, a cirurgia não terá sucesso, podendo causar catástrofes. O mesmo vale para outras aplicações, como servomecanismos, controle de processos e etc.

2.3 Técnicas de Projeto de Controle para Sistemas em Tempo Discreto

Quando se projeta controladores para sistemas em tempo discreto, existem muitas técnicas para realizar o projeto. Muita das vezes, precisa-se fazer uma análise minuciosa da técnica a ser emprega, devido ao teor do projeto em questão. A seguir, mostram-se algumas das técnicas utilizadas atualmente e que mais tarde será feito uma comparação das mesmas com a técnica que este trabalho aborda.

2.3.1 Alocação de Polos

Um dos métodos de se computar o ganho de realimentação de estado para alocação de autovalores ou polos. O método, no entanto, tem a restrição de que os autovalores selecionados não podem conter nenhum autovalor da matriz **A**.

Considere (**A**, **B**) controlável, onde **A** é uma matriz $n \times n$ e **B** é $n \times 1$. O objetivo é encontrar uma matriz **K** $1 \times n$ tal que (**A**, **BK**) tenha qualquer conjunto de autovalores desejados que não fazem parte dos autovalores de **A**. O procedimento para isso é descrito a seguir.

Dada uma matriz **F** da forma:

$$\mathbf{F} = \begin{bmatrix} \lambda_1 & 0 & 0 & 0 & 0 \\ 0 & \alpha_1 & \beta_1 & 0 & 0 \\ 0 & -\beta_1 & \alpha_1 & 0 & 0 \\ 0 & 0 & 0 & \alpha_2 & \beta_2 \\ 0 & 0 & 0 & -\beta_2 & \alpha_2 \end{bmatrix} \tag{2.12}$$

onde λ_i representa o i-ésimo autovalor real desejado, e α_i e β_i são a parte real e imaginária, respectivamente, do i-ésimo autovalor complexo desejado.

Para se encontrar o controlador que posiciona o sistema nos autovalores desejados, pode-se seguir o seguinte procedimento.

a) Selecione uma matriz **F** $n \times n$ que tenha o conjunto de autovalores desejados.

b) Selecione um vetor $\bar{\mathbf{K}}$ $1 \times n$ tal que (**F**, $\bar{\mathbf{K}}$) é observável.

c) Resolva a equação de Lyapunov $\mathbf{AT} - \mathbf{TF} = \mathbf{B}\bar{\mathbf{K}}$ em função de $\mathbf{T}$ o qual tem uma única solução.

d) Calcule a matriz de ganho $\mathbf{K} = \bar{\mathbf{K}}\mathbf{T}^{-1}$.

2.3.2 Regulador Quadrático Linear

O regulador quadrático linear (*linear quadratic regulator* - LQR) da teoria de controle ótimo pode ser usado para resolver muitos problemas de projeto do controlador em que o estado é acessível, a regulação e o esforço do atuador são medidos por um desvio médio quadrático. Uma formulação estocástica do problema LQR é conveniente; uma formulação mais usual é como um problema de controle ótimo. O sistema é descrito por [1]

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} + \mathbf{w},$$

onde w é um ruído branco de média zero, isto é, w tem matriz de densidade espectral de potência $S_{\mathrm{w}}(w) = I$ para todos w. O estado $\mathbf{x}$ está disponível para o controlador, então $\mathbf{y} = \mathbf{x}$ nesse *framework*.

A função de custo do LQR é a soma do estado ponderado do quadrado médio do estado estacionário $\mathbf{x}$, e o sinal do atuador ponderado do quadrado médio do estado estacionário $\mathbf{u}$ [1]

$$J_{\mathrm{lqr}} = \lim_{t\to\infty} \mathbf{E}(\mathbf{x}(t)^T Q\mathbf{x}(t) + \mathbf{u}(t)^T R\mathbf{u}(t)), \tag{2.13}$$

onde Q e R são matrizes de peso semi-definido positivos. O primeiro termo penaliza as derivadas de $\mathbf{x}$ de zero, e o segundo termo representa o custo de usar o sinal do atuador. Pode-se expressar esse custo, criando o sinal de saída regulado $\begin{bmatrix} R^{\frac{1}{2}}\mathbf{u} & Q^{\frac{1}{2}}\mathbf{x} \end{bmatrix}'$, portanto, $J_{\mathrm{lqr}} = \lim_{t\to\infty} \mathbf{E}z(t)^T z(t)$, o desvio quadrático médio de z. Como w é um ruído branco, tem-se $J_{\mathrm{lqr}} = ||H||_2^2$, o quadrado da norma $\mathbf{H}_2$ da matriz de transferência em malha fechada.

A planta para o problema de regulador LQR é dada por: $\mathbf{A}_{\mathrm{p}} = \mathbf{A}$, $\mathbf{B}_{\mathrm{u}} = \mathbf{B}$, $\mathbf{B}_{\mathrm{w}} = \mathbf{I}$, $\mathbf{C}_{\mathrm{z}} = [0\ Q^{\frac{1}{2}}]'$, $\mathbf{C}_{\mathrm{y}} = \mathbf{I}$, $\mathbf{D}_{\mathrm{zw}} = \mathbf{0}$, $\mathbf{D}_{\mathrm{zu}} = [R^{\frac{1}{2}}\ 0]'$, $\mathbf{D}_{\mathrm{yw}} = \mathbf{0}$ e $\mathbf{D}_{\mathrm{yu}} = \mathbf{0}$, onde:

$$\begin{cases} \dot{\mathbf{x}} = \mathbf{A}_{\mathrm{p}}\mathbf{x} + \mathbf{B}_{\mathrm{w}}\mathbf{w} + \mathbf{B}_{\mathrm{u}}\mathbf{u} \\ \mathbf{z} = \mathbf{C}_{\mathrm{z}}\mathbf{x} + \mathbf{D}_{\mathrm{zw}}\mathbf{w} + \mathbf{D}_{\mathrm{zu}}\mathbf{u} \\ \mathbf{y} = \mathbf{C}_{\mathrm{y}}\mathbf{x} + \mathbf{D}_{\mathrm{yw}}\mathbf{w} + \mathbf{D}_{\mathrm{yu}}\mathbf{u} \end{cases}, \tag{2.14}$$

As especificações consideradas são a capacidade de realização e a especificação de desigualdade funcional $||H||_2 \leq \alpha$. As suposições-padrão são que (Q,A) é observável, (A,B) é controlável, e $R > 0$, caso em que a especificação declarada na última sentença é mais forte do que (*i.e.*, implica) estabilidade interna. Com estas suposições-padrão, existe, na verdade um controlador que atinge o menor custo possível de LQR, e é uma realimentação de estado constante, $K_{\text{lqr}}(s) = -K_{\text{sfb}}$, que pode ser encontrado com [1]

Dado que X_{lqr} tenha única solução positiva definida da equação algébrica de Riccati [60] que pode ser encontrada através da matriz Hamiltoniana associada M [61], e depois dada uma matriz T, tal que:

$$T^{-1}MT = \begin{bmatrix} \hat{A}_{11} & \hat{A}_{12} \\ 0 & \hat{A}_{22} \end{bmatrix},$$

onde $\hat{A}_{11}$ é estável, pode-se particionar T como

$$T = \begin{bmatrix} T_{11} & T_{12} \\ T_{21} & T_{22} \end{bmatrix},$$

Portanto, a solução $X_{\text{lqr}} = T_{21}T_{11}^{-1}$. Logo $K_{\text{lqr}} = R^{-1}B^T X_{\text{lqr}}$, com isso pode-se chegar a $J_{\text{lqr}} = \mathbf{Tr}X_{\text{lqr}}$. Em particular, a especificação $||H||_2 \leq \alpha$ (e também de realização) é alcançável se e somente se $\alpha \geq \sqrt{\mathbf{Tr}X_{\text{lqr}}}$, caso em que o controlador LQR ótimo K_{lqr} obedeça às especificações.

2.3.3 Síntese Indutiva Guidada por Contraexemplos - CEGIS

CEGIS é um processo iterativo para a síntese de programas e atualmente está se tornando popular. Cada iteração realiza uma generalização indutiva baseada em contraexemplos fornecida por um mecanismo de verificação. Normalmente, a generalização indutiva usa informações sobre um número limitado de entradas para fazer afirmações sobre todas as possíveis entradas, na forma de soluções candidatas [23].

A Figura 2.5 mostra a estrutura do CEGIS [23] e consiste em duas etapas principais: síntese e verificação. A etapa de verificação consiste em verificar a satisfabilidade dos requerimentos em relação a uma determinada propriedade analisada, caso ela falhe, ou seja bem-sucedida. O estágio de síntese, por sua vez, consiste em gerar um programa candidato, de modo a verificar se ele satisfaz uma propriedade analisada.

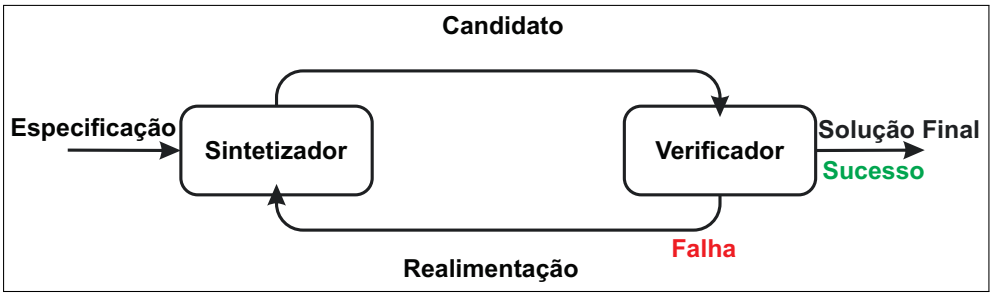


Figura 2.5: Diagrama em Blocos CEGIS.

No estágio de síntese, há um algoritmo que aprende com contraexemplos fornecidos por um mecanismo de verificação. O algoritmo de aprendizagem procede pesquisando o espaço de candidatos conceitos para encontrar um que seja consistente com os exemplos vistos até o momento. Pode haver vários conceitos consistentes, e a estratégia de busca determina o candidato escolhido. Esse candidato é então apresentado ao mecanismo de verificação, que verifica o candidato quanto à especificação de correção. Se o candidato estiver correto, o sintetizador finaliza e produz este candidato. Caso contrário, o mecanismo de verificação gera um contraexemplo, gerando o motivo da falha. Esse contraexemplo é retornado ao algoritmo de aprendizagem, que adiciona o contraexemplo ao seu conjunto de exemplos e repete sua pesquisa. É possível que, após algum número de iterações desse *loop*, o algoritmo de aprendizagem possa ser incapaz de encontrar um candidato consistente; nesse caso, a etapa de aprendizagem e, portanto, o procedimento CEGIS, falha [62].

Dada a especificação do programa desejado, σ, o procedimento de síntese indutiva gera um programa candidato P^* que satisfaz $\sigma(P^*, \vec{x})$ para um subconjunto $\vec{x}_{inpts}$ de todas as entradas possíveis. O programa candidato P^* é passado para o estágio de verificação, que verifica se ele satisfaz a especificação $\sigma(P^*, \vec{x})$ para todas as entradas possíveis. A técnica faz isso verificando se $\neg\sigma(P^*, \vec{x})$ é insatisfatório. Se assim for, $\forall\vec{x}.\neg\sigma(P^*, \vec{x})$ é válido, e foi sintetizado com sucesso uma solução e o algoritmo é finalizado. Caso contrário, o verificador produz um contraexemplo $\vec{c}$ da atribuição satisfatória, que é então adicionada ao conjunto de entradas passadas para o sintetizador, e o *loop* é repetido.

O método utilizado nos blocos de síntese e verificação varia em diferentes implementações do CEGIS. As principais contribuições deste trabalho estão concentradas na Figura 2.5, onde será mostrado nos capítulos posteriores como é feito a verificação e a síntese.

2.4 Implementação de Controladores Digitais

A implementação das leis de controle usando um computador possui alguns problemas e dificuldades. O principal problema é implementar um sistema de tempo discreto. Os princípios para fazer isso foram abordados em detalhes. É simples gerar o código do algoritmo de controle. Filtragem digital não-linear sofisticada para remoção de valores discrepantes também será discutida. O atraso computacional é influenciado consideravelmente pela organização do código de computador. Dificuldades que surgem da saturação em atuadores e formas de evitar essas dificuldades são discutidas. Isso também fornece automaticamente uma solução para a troca e inicialização de modo de operação.

Os métodos de projeto baseados em alocação de polos por realimentação de estado fornecem um controlador da forma [56]

$$\begin{aligned} \mathbf{x}(k+1) &= \mathbf{F}\mathbf{x}(k) + \mathbf{G}\mathbf{y}(k) + \mathbf{G}_c\mathbf{u}_c(k) \\ \mathbf{u}(k) &= \mathbf{C}\mathbf{x}(k) + \mathbf{D}\mathbf{y}(k) + \mathbf{D}_c\mathbf{u}_c(k) \end{aligned} \tag{2.15}$$

A implementação de um sistema de tempo discreto descrito pela equação (2.15) usando um computador digital é simples. Os detalhes dependem do *hardware* e *software* disponíveis. Para mostrar os princípios, assume-se que o sistema descrito pela equação (2.15) deve ser implementado usando um computador digital com conversores ADC e DAC e um *clock* de tempo real. Uma representação gráfica do programa é mostrada na Figura 2.6. A execução do programa é controlada pelo *clock*. A barra horizontal indica que a execução é interrompida até que uma interrupção venha do *clock*. O *clock* é ajustado para que uma interrupção seja obtida em cada instante de amostragem. O código no bloco é executado após cada interrupção.

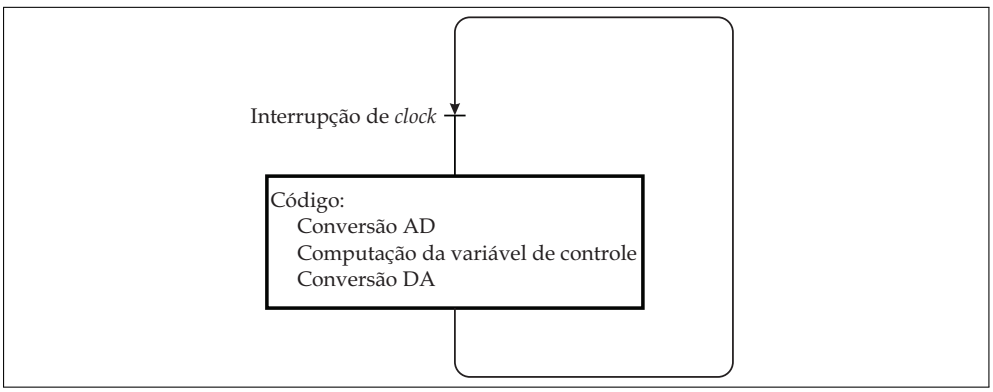


Figura 2.6: Representações gráficas de um programa usado para implementar um sistema de tempo discreto.

O corpo do código é dado na Figura 2.7. A conversão analógico-digital é comandada na primeira linha. Os valores apropriados são armazenados nas matrizes `y` e `uc`. O sinal de controle **u** é calculado na segunda linha usando multiplicação vetor-matriz e adição vetorial. O vetor de estado **x** é atualizado na terceira linha e a conversão digital-analógico é realizada na quarta linha. Para obter um código completo, também é necessário ter declarações de tipo para os vetores `u`, `uc`, `x` e `y`, e as matrizes `F`, `G`, `Gc`, `C`, `D` e `Dc`. Também é necessário atribuir valores às matrizes e o valor inicial do estado **x**. Ao usar linguagens de computador que não possuem operações de matriz, é necessário escrever procedimentos apropriados para gerar operações de matriz usando operações em escalares. Observe que as segunda e terceira linhas do código correspondem exatamente à equação (2.15).

```
Procedimento Regular()
inicio
1 Adin y uc
2 u=C*x+D*y+Dc*uc
3 x=F*x+G*y+Gg*uc
4 Daout u
fim
```

Figura 2.7: Esqueleto de código de computador para a lei de controle na equação (2.15).

Para obter um bom sistema de controle, também é necessário considerar os seguintes fatores:

- Pré-filtragem e atraso computacional: Para evitar o *aliasing*, é necessário usar um pré-filtro analógico para eliminar perturbações com frequências maiores que a frequência de Nyquist associada à taxa de amostragem [55]. Em um problema de controle, normalmente há muito mais informação disponível sobre os sinais como equações diferenciais para os modelos de processo e possivelmente também para as perturbações. Geralmente, é útil amostrar os sinais analógicos a uma taxa comparativamente alta e evitar o *aliasing* por um pré-filtro analógico comum projetado do ponto de vista do processamento do sinal, visto que o número de *bits* influencia a relação sinal ruido (*signal to noise ratio* - SNR) e isso deixa o sinal mais fiel. Como o pré-filtro analógico tem dinâmica, é necessário incluir a dinâmica de filtro no modelo de processo. Se o pré-filtro ou a taxa de amostragem for alterada, os coeficientes da lei de controle devem ser recalculados. Com taxas de amostragem normais, isto é, 15 a 45 vezes por período, é necessário considerar

a dinâmica do pré-filtro no projeto do controlador. Como as conversões e cálculos AD e DA levam tempo, sempre haverá um atraso quando uma lei de controle for implementada usando um computador. O atraso, que é chamado de atraso computacional, depende de como o algoritmo de controle é implementado [56]. Existem basicamente duas maneiras diferentes de fazer isso. Na primeira, as variáveis medidas lidas no tempo t_k podem ser usadas para calcular o sinal de controle a ser aplicado no tempo t_{k+1}. Outra possibilidade é ler as variáveis medidas no tempo t_k e fazer a conversão DA o mais rápido possível.

- Não-linearidades do(s) atuador(s): A teoria linear tem uma ampla aplicabilidade, muitas vezes existem algumas não-linearidades que devem ser consideradas. Por exemplo, acontece frequentemente que os atuadores são não-lineares. As válvulas são comumente usadas como atuadores em sistemas de controle de processo. Isto corresponde a uma não-linearidade do tipo saturação, onde os limites correspondem a uma válvula totalmente aberta ou fechada. A não-linearidade é, portanto, importante quando grandes mudanças são feitas. Pode haver dificuldades com o sistema de controle durante a inicialização e o desligamento, bem como durante grandes mudanças, se as não-linearidades não forem consideradas. Um exemplo típico é a integração do integrador. Outras não-linearidades típicas em sistemas do mundo real são limitações de taxa, histerese e *backslash* [63]. A maneira racional de lidar com a saturação é desenvolver uma teoria de projeto que considere a não-linearidade. Isso pode ser feito usando a teoria de controle ótimo [64]. No entanto, tal método de projeto é bastante complicado. A lei de controle correspondente também é complexa. Portanto, é prático usar métodos heurísticos simples.

- Aspectos operacionais: Entre o controlador e o operador do sistema de controle existem muitas outras coisas. Isso inclui uma avaliação das informações exibidas ao operador e os mecanismos para o operador alterar os parâmetros do controlador. Para discutir a interface do operador, é necessário considerar como o sistema será utilizado operacionalmente. Primeiro, é importante perceber a ampla variedade de aplicações dos sistemas de controle. Não há como dar um tratamento abrangente. Por exemplo, as demandas são muito diferentes para um piloto automático, uma sala de controle de processos ou uma planta de uma nave espacial. Muitas vezes é desejável ter a possibilidade de executar um sistema controle manualmente. Como o controlador é um sistema dinâmico, o estado do controlador deve ter o valor correto quando o modo é alternado de manual

para automático. Se este não for o caso, existe um modo de rastreamento, que ajusta o estado do controlador para que seja compatível com as entradas e saídas do controlador. Um modo de rastreamento pode ser visto como uma implementação de um observador. Com o controle digital, é possível ter muitos outros modos de operação. Estimativa de parâmetros e algoritmos de projeto de controle podem ser incluídos no controlador. Com um controlador sendo um sistema dinâmico, é importante definir o estado do controlador apropriadamente quando o controlador estiver ligado. Com o controle digital, é possível usar uma parametrização no algoritmo de controle e outra na comunicação do operador. Os parâmetros exibidos para o operador podem então estar relacionados ao desempenho do sistema e não aos detalhes do algoritmo de controle. A conversão entre os parâmetros é feita por um algoritmo no computador. É muito importante certificar-se de que um sistema de controle digital funcione com segurança. Idealmente, isso significa que o sistema deve fornecer o resultado correto ou um alarme se não estiver funcionando corretamente [56].

- Realização: Ao longo deste capítulo, será mostrado como o arredondamento e a quantização nos conversores ADC e DAC influenciam o comportamento do sistema. Os erros de arredondamento nos cálculos da lei de controle também causam quantização, que pode ser modelada e analisada da mesma forma que a quantização do conversor. A quantização decorrente das computações depende criticamente de como as computações são organizadas, por exemplo, sobre como o controlador de dados amostrados é realizado. Algumas realizações diferentes são: forma direta, forma companheira, forma em série (Jordan), forma paralela (diagonal), forma de escada e a forma operador δ [56]. A realização também tem o poder de influenciar o comportamento sob os efeitos de FWL.

- Aspectos de programação: Praticamente todos os controladores de tempo discreto são implementados em um sistema operacional de tempo real [56]. Em alguns sistemas, as diferentes partes dos algoritmos podem ser distribuídas entre diferentes processadores. A comunicação pode introduzir atrasos variáveis no tempo (*jitter*) no período de amostragem. A programação é um aspecto importante da implementação de um sistema de controle, tanto no que diz respeito à eficiência do sistema quanto ao tempo necessário para a implementação. O esforço necessário e as abordagens utilizadas dependem do *software* disponível e da natureza do problema de controle. O código é normalmente

escrito em C ou C++. Ada [65], desenvolvida pelo Departamento de Defesa dos EUA para aplicações de controle digital, é a primeira linguagem projetada e desenvolvida para aplicações de tempo real. O caráter e a dificuldade da programação dependem muito da aplicação. Os requisitos sobre as comunicações do operador são críticos. O código necessário para a comunicação do operador é geralmente muito maior que o código de controle puro. A Figura 2.8 mostra um exemplo de código embarcado no computador digital para operar um sistema de controle, onde o procedimento `Regular` é o código necessário para implementar o algoritmo de controle desejado e `Display` calcula algumas variáveis e as exibe em formato analógico ou digital.

```
1 repita
2 Espere por uma interrupção do clock
3 Regular()
4 Display()
5 sempre
```

Figura 2.8: Esqueleto de código de computador para um *loop* de controle.

2.4.1 Efeitos de Tamanho de Palavra Finita

Outra importante propriedade a se considerar na implementação em controle digital é o que a literatura chama de efeitos de palavra finita (*finite word length* - FWL) [66]. Devido ao foco de este trabalho considerar essa propriedade, que muitas vezes é deixada de lado, um pouco mais de detalhe é mostrado a seguir.

Ao implementar um sistema de controle digital, é necessário verificar alguns fatores, tais como: precisão dos conversores, precisão necessária nos cálculos, o método de cálculo (em aritmética de ponto fixo ou ponto flutuante), etc. Com isso, deve-se entender os efeitos das limitações e estimar suas consequências para o sistema de malha fechada. Esta não é uma questão trivial, porque a resposta depende de uma interação complexa de realimentação, do algoritmo e da taxa de amostragem. Felizmente, apenas estimativas brutas devem ser feitas. Por exemplo, a quantidade de *bits* de resolução (*e.g.*, 10 ou 12 *bits*) e o comprimento da palavra (*e.g.*, 24 ou 32 *bits*).

Algoritmos de controle digital são tipicamente implementados em microcontroladores e microprocessadores, que possuem comprimentos de 8, 16 ou 32 *bits*. Para ilustrar o efeito

que o tamanho da palavra podem ocasionar na computação numérica, usa-se o produto escalar dos vetores $a = [100\ 1\ 100]$ e $b = [100\ 1\ -100]$. O produto escalar é $(a,b) = 1$. Se o produto escalar for calculado em representação de ponto flutuante com uma precisão correspondente a três casas decimais, o resultado será zero porque $100 \times 100 + 1 \times 1$ é arredondado para 10000. Observe que o resultado obtido depende da ordem das operações. As operações de comprimento de palavra finita não são associativas nem distributivas.

As dificuldades de implementação podem ser evitadas sem usar o cálculo completo de precisão dupla, adicionando os termos em precisão dupla e arredondando para precisão simples depois. Esse método pode ser aplicado a cálculos de ponto fixo e de ponto flutuante. Observe que a instrução de multiplicação, para muitos microprocessadores, é implementada para que a operação de multiplicação esteja disponível com precisão dupla. Muitas linguagens de alto nível também possuem construções que suportam esse tipo de cálculo. De modo geral, arredondamento e quantização darão origem a pequenos erros, ao passo que os efeitos do *overflow* [67] serão desastrosos.

Outra análise que pode ser feita sobre os efeitos FWL em sistemas em tempo discreto com realimentação de estados é a seguinte:

A matriz do controlador de realimentação de estado estático K de um sistema digital como da equação (2.4) é fortemente afetada pelos efeitos de FWL, que podem comprometer algumas propriedades do sistema e devem ser manipuladas durante as fases de projeto.

Os efeitos da FWL nos coeficientes do controlador são descritos da seguinte forma:

$$\mathscr{FWL}_{\langle I,F\rangle}[\cdot] : \mathbb{R}^{m\times n} \to \mathbb{R}_Q^{m\times n}\ , \tag{2.16}$$

onde $\mathbb{R}_Q^{m\times n}$ é o conjunto discreto de matrizes $m \times n$ composto pelos elementos $\mathbb{R}^{m\times n}$, que podem ser representados no formato de ponto fixo $\langle I,F\rangle$ e I e F são o número de *bits* das partes inteiras e fracionárias, respectivamente.

Supondo que apenas K é digital e implementado com aritmética de ponto fixo, cujos coeficientes estão sujeitos a efeitos FWL, um sistema dinâmico diferente de espaço de estados digital de ordem n com controlador de realimentação de estado Ω_{fwl} similar a da equação (2.4) tal como o sinal de controle é $\mathbf{u}(k) = \mathbf{r}(k) - \mathbf{K}_{\text{fwl}}\mathbf{x}(k)$, onde a matriz $\mathbf{K}_{fwl} = \mathscr{FWL}_{\langle I;F\rangle}[\mathbf{K}]$ é a matriz do controlador que sofre efeitos de FWL. Como consequência, tais efeitos de FWL podem influenciar os parâmetros de resposta a degraus do sistema, podendo não atender às

especificações.

2.4.2 Algoritmo Genético

O algoritmo genético é um método para resolver problemas de otimização restritos e não restritos que é baseado na seleção natural, o processo que impulsiona a evolução biológica. O algoritmo genético modifica repetidamente uma população de soluções individuais [25]. Em cada etapa, o algoritmo genético seleciona os indivíduos aleatoriamente da população atual para serem pais e os utiliza para produzir filhos para a próxima geração. Ao longo de gerações sucessivas, a população "evolui"em direção a uma solução ótima. O algoritmo genético pode ser aplicado para resolver uma variedade de problemas de otimização que não são adequados para algoritmos de otimização padrão, incluindo problemas nos quais a função objetivo é descontínua, não diferenciável, estocástica ou altamente não-linear. O algoritmo genético pode resolver problemas de programação inteira mista, onde alguns componentes são restritos a valores inteiros.

O algoritmo genético usa três tipos principais de regras em cada etapa para criar a próxima geração da população atual:

- As regras de seleção selecionam os indivíduos, chamados pais, que contribuem para a população na próxima geração;
- As regras de *crossover* combinam dois pais para formar filhos para a próxima geração;
- As regras de mutação aplicam alterações aleatórias a pais individuais para formar filhos.

No presente trabalho, o algoritmo genético será utilizado simplesmente como ferramenta aplicada para geração de controladores candidatos, no processo de síntese.

2.5 Resumo

Neste capítulo, foram introduzidos os conceitos básicos para o entendimento desta dissertação, relacionadas a basicamente síntese de controladores digitais através de métodos formais e das técnicas tradicionais de sistemas de controle. Mais especificamente, explicaram-se conceitos básicos de verificação formal e suas técnicas mais utilizadas, com o objetivo de se

chegar em na aplicação do conceito de invariantes. Também foram mostrados alguns conceitos sobre representações de sistemas dinâmicos lineares. Outro ponto que foi discutido, foi a respeito das técnicas de projeto de controle para sistemas em tempo discreto. Também foi apresentado uma introdução a forma como controladores digitais são implementados em um microprocessador e também a respeito de efeitos FWL nesses controladores. Por fim, foi relatado o conceito básico de algoritmo genético. Como resultado, o conteúdo deste capítulo fornece todo o embasamento necessário para compreensão do trabalho desenvolvido, que será descrito nas seções seguintes.

Capítulo 3

Trabalhos Relacionados

Este capítulo busca apresentar os trabalhos direta ou indiretamente relacionados a presente dissertação e as suas subseções seguintes apresentam os trabalho focados em verificação (cf. Seção 3.1) e síntese (cf. Seção 3.2) de sistemas de controle. Ao final do capítulo é mostrado uma comparação do trabalho proposto e os trabalhos correlatos.

O presente trabalho tem duas frentes de atuação que se relacionam. A primeira, de atuação é a verificação formal de sistemas de controle, onde considera-se aspectos de desempenho (tempo de assentamento e sobressinal) e de implementação (efeitos de FWL). A outra trata da síntese de controladores, considerando esses mesmo aspectos de desempenho e de implementação, onde a *engine* de síntese utiliza a verificação dessas propriedades de desempenho em seu mecanismo que neste trabalho é baseado na técnica de CEGIS.

3.1 Verificação

Existem alguns trabalhos que tem por finalidade abordar os problemas relacionados à verificação de requisitos de sistemas de controle, seja em tempo contínuo ou discreto. Contudo, como já mencionado anteriormente, esta dissertação se direciona à sistemas em tempo discreto. Há alguns trabalhos que lidam com a verificação de diferentes requisitos de sistemas de controle, porém, a presente pesquisa tem como objetivo tratar da verificação de requisitos de desempenhos de sistemas de controle.

Bessa *et al.* [6] apresentam uma metodologia de verificação para determinar formalmente a estabilidade incerta de sistemas lineares em controladores digitais com considerações

sobre os aspectos de implementação. Especificamente, esta estratégia é combinada com o DSVerifier, que é uma ferramenta de verificação que utiliza a verificação de modelo limitado com base nas teorias de módulo de satisfabilidade (*satisfiability modulo theories* - SMT) para verificar a estabilidade dos sistemas de controle digital considerando incerteza. O DSVerifier determina a estabilidade do sistema de controle, considerando a planta, juntamente com os efeitos do comprimento da palavra finita (*finite word length* - FWL) na implementação do controlador digital. Em seguida, verifica-se a estabilidade robusta "não-frágil" de um determinado sistema em malha fechada. A metodologia proposta e a respectiva ferramenta (DSVerifier) são avaliadas considerando exemplos de controle não frágeis encontrados na literatura. O trabalho mostra que a técnica é capaz de prever problemas de fragilidade em controladores robustos quando considera-se a estabilidades de sistemas de controle digital, visto que considera os efeitos FWL. No entanto, este trabalho se concentra em apenas verificar a estabilidade, sendo que aspectos de desempenho não são considerados, visto que as propriedades de desempenho são muito importantes quando se projeta controladores, além de já considerarem a estabilidade.

Wang *et al.* [68] apresentam uma abordagem de verificação da robustez de sistemas de controle, a nível de código e modelagem, onde se baseia em um cálculo invariante na dinâmica discreta do sistema. Usando solucionadores de programação semi-definida (*semidefinite programming* - SDP), uma função baseada em Lyapunov é sintetizada, capturando as margens do vetor do sistema linear em malha fechada considerado. Essa invariante numérica expressa sobre as variáveis de estado do sistema é compatível com a análise de código e permite sua validação no artefato de código. Esta análise automática amplia as técnicas de verificação focadas na implementação do controlador, abordando a validação da robustez no modelo e no nível de código. Ele foi implementado em uma ferramenta que analisa sistemas SISO discretos e gera super-aproximações de margens de fase e ganho. Os autores apresentam uma abordagem eficaz para verificar robustez dos controladores digitais a nível de código e modelagem. Da mesma forma, esse trabalho não considera aspectos de desempenho do controlador, apesar de considerar aspectos relativos a efeitos causados pela aritmética de ponto fixo que se enquadra como um aspecto de implementação de controladores.

Sadraddini e Belta [69] desenvolveram um método para controlar sistemas de tempo discreto com parâmetros constantes, mas inicialmente desconhecidos a partir de especificações de lógica temporal linear (*linear temporal logic* - LTL). Os autores usam as noções de sistemas de transição paramétricos e adaptativos (não-determinísticos) e ferramentas de métodos formais

para computar estratégias de controle adaptativo para sistemas finitos. Apesar de não utilizar os métodos tradicionais de controle adaptativo, usando uma abordagem correta por construção, a qual não requer um modelo de referência e pode lidar com uma gama muito maior de sistemas e especificações, não leva em consideração aspectos relativos a desempenho de sistemas considerando a resposta ao degrau, bem como deixa de lado aspectos enfrentados com a implementação, como os efeitos de FWL. Como a maioria das outras aplicações de métodos formais, os resultados sofrem pela alta complexidade computacional. Como discutido no artigo, o número de estados no sistema de transição adaptativa (*adaptative transition system* - ATS) pode ser muito grande. Além disso, a construção de quocientes finitos para sistemas infinitos é computacionalmente difícil.

Gross *et al.* [17] propuseram uma abordagem para formalizar requisitos comuns de sistema de controle de atitude de naves espaciais, como limites do atuador, erro de indicação, alcançabilidade, desvio, tempo de assentamento, tempo de subida e sobressinal. Os autores utilizaram métodos formais, mais especificamente verificação de modelos e teste de hipótese para verificar requisitos de desempenho de sistemas de controle de atitude de naves espaciais. O trabalho em questão avalia esses sistemas e verifica se obedecem a esses requisitos no domínio do tempo contínuo. Embora a intenção inicial, de formalizar os requisitos, fosse conduzir análises formais de métodos para determinar se o projeto de controle de atitude de naves espaciais atendia aos requisitos do sistema, as equações de movimentos não-lineares se mostraram muito complexas para os solucionadores e apenas alguns requisitos puderam ser analisados. Um ponto positivo desse trabalho foi que consideraram aspectos de desempenho na verificação de controladores. Apesar disso, os autores não trabalharam com sistemas em tempo discreto, não levando em consideração aspectos de implementação, como os efeitos FWL.

Hassani e Lee [70] apresentam um paradigma de projeto genérico multi-objetivo que utiliza otimização de enxame de particulas (*particle swarm optimization* - PSO) com comportamento quântico (QPSO) para decidir a configuração ideal do controlador LQR para um determinado problema, considerando um conjunto de objetivos concorrentes. Existem três contribuições principais introduzidas nesse trabalho da seguinte forma: o algoritmo QPSO padrão é reforçado com um esquema de inicialização informado baseado no algoritmo de recozimento simulado e no mecanismo de seleção de vizinhança gaussiana, ele também é aumentado com uma estratégia de busca local que integra as vantagens do algoritmo memético no QPSO convencional, e também é introduzido um critério agregado de ponderação dinâmica que combina

dinamicamente as restrições *soft* e *hard* com os objetivos de controle para fornecer ao projetista um conjunto de soluções ótimas de Pareto e permite que ela decida a solução de destino com base nas preferências práticas. O trabalho em questão se mostra bastante promissor, visto que faz uma certa análise a aspectos de desempenho de sistemas de controle. Apesar disso, a análise toda é feita considerando controladores em tempo contínuo. Apenas é feita uma "melhoria" nos parâmetros de desempenho do sistema, mas não sintetiza controladores com base nesses parâmetros como requisitos necessários e específicos. Como não trabalha diretamente com sistemas de controle digital, consequentemente não leva em conta aspectos de implementação dos mesmo em microprocessadores, podendo gerar controladores que quando implementados nessas plataformas não se comporte conforme projetado, visto que podem sofrer com os efeitos de FWL, por exemplo.

Em sistemas dinâmicos, requisitos de desempenho são propriedades funcionais que se violadas, podem acarretar danos irreparáveis. Por exemplo, em sistemas de navegação de mísseis, caso esses requisitos funcionais não sejam atendidos uma catástrofe pode acontecer, são sistemas bem críticos, onde não se pode correr riscos. O requisitos de desempenho abordados neste trabalho são o tempo de assentamento e sobressinal que são propriedades correlacionadas e ligadas a resposta do sistema.

A presente dissertação visa alguns aspectos que muitos desses trabalhos abordam mas nenhum deles cumpre todos esses aspectos ao mesmo tempo, tais como análise em sistemas de tempo discreto, requisitos de desempenho de sistemas de controle (tempo de assentamento e sobressinal) e consideração de aspectos de implementação (efeitos de FWL).

3.2 Síntese

No contexto de síntese de sistemas de controle, alguns trabalhos que desenvolvem esse processo em sistemas de controle foram investigados visando extrair aspectos importantes de cada um e mostrando que a presente dissertação busca preencher lacunas que esses trabalhos deixam.

Ravanbakhsh e Sankaranarayanan [71] investigam o problema de sintetizar controladores robustos que asseguram que o sistema em malha fechada satisfaça uma especificação de *reach-while-stay* de entrada, em que todas as trajetórias partindo de um conjunto inicial I, eventualmente atingem um conjunto de metas especificadas G, enquanto permanecem dentro de um

conjunto seguro *S*. A ideia chave é: dado o modelo da planta que consiste em um sistema comutado de tempo contínuo controlado por um sinal de comutação externo e entradas de perturbação da planta, o controlador usa uma lei de realimentação de estado para controlar o sinal de comutação, a fim de garantir que as propriedades de correção desejadas sejam mantidas, independentemente das ações de perturbação. Para garantir a especificação de *reach-while-stay*, a abordagem usa um certificado de prova na forma de uma função robusta de controle Lyapunov (*robust control Lyapunov-like function* - RCLF). Para encontrar um RCLF, usa-se uma estrutura de CEGIS, resolvendo iterativamente uma fórmula $\exists\forall\exists\forall$ usando solucionadores SMT livres de quantificação. Finalmente, utilizaram o problema de traduzir o RCLF sintetizado pela abordagem em uma implementação de controle. A técnica de síntese de controladores para sistemas de controle comutados apresentou bons resultados pelo que se propôs a fazer, considerando sistemas de controle em tempo contínuo. Porém, deixou de considerar aspectos de implementação desses controladores do mundo real, onde necessita-se trabalhar com sistemas em tempo discreto, e levar em conta aspectos de implementação, como efeitos de FWL. O problema de sintetizar controladores para satisfazer requisitos da propriedade *reach-while-stay* tem sua importância, mas outros aspectos como estabilidade e requisitos de desempenham devem ser considerados em projeto de controladores para problemas do mundo real.

Abate *et al.* [72–74] propõem uma abordagem sólida e automatizada para sintetizar controladores digitais seguros com realimentação para plantas físicas representadas como modelos lineares invariantes no tempo. Os autores utilizaram CEGIS o qual possui duas fases: síntese de um controlador de realimentação estático que estabiliza o sistema, mas que pode não ser seguro para todas as condições iniciais. A segurança é então verificada via BMC ou aceleração abstrata; se a etapa de verificação falhar, um contraexemplo é fornecido ao mecanismo de síntese e o processo itera até que um controlador seguro seja obtido. O trabalho em questão apresenta excelentes resultados, porém, ainda deixa de considerar aspectos de desempenho de projeto de controladores, considerando somente a estabilidade dos mesmos. Um bom aspecto é que a metodologia considera fragilidade na implementação dos controladores digitais.

Yordanov *et al.* [8] apresentam uma estrutura computacional para síntese automática de uma estratégia de controle de realimentação para um sistema de tempo discreto (mais precisamente sistemas PWA) a partir de uma especificação dada como uma lógica linear temporal (*linear temporal logic* - LTL) sobre um conjunto arbitrário de predicados lineares nas variáveis de estado do sistema. A abordagem consiste em definir partições apropriadas para seu estado e

espaços de entrada, e então constrói-se uma abstração finita do sistema na forma de um sistema de transição de controle. Depois disso, aproveitando ideias e técnicas de verificação de modelos LTL e jogos de Rabin, desenvolveram um algoritmo para gerar uma estratégia de controle para a abstração finita. A abordagem apresentada para a construção da abstração garante que uma estratégia de controle gerada para o sistema de controle finito pode ser facilmente transformada em uma estratégia de controle para o sistema PWA inicial. Embora comprovadamente correta, a solução geral é conservadora e computacionalmente cara. Apesar desta abordagem conseguir sintetizar controladores, deixa de considerar alguns aspectos bem relevantes em projetos de controladores digitais, como aspectos de desempenho e efeitos da fragilidade de controladores quando implementados em microprocessadores.

Nilsson *et al.* [14] descrevem duas abordagens para sintetizar controladores corretos por construção em sistemas de controle de cruzeiro adaptativo. Ambas as abordagens baseiam-se na computação em pontos fixos no domínio do controlador, a partir da qual a especificação em LTL pode ser aplicada. Uma calcula tais pontos fixos diretamente no espaço de estados contínuo, a outra em uma abstração de estados finitos da dinâmica não-linear. De acordo com os resultados experimentais, este trabalho se mostra eficaz para sistemas de controle de cruzeiro adaptativo. Porém, considera a síntese de controladores estáveis considerando a propriedade de *safety*, mas não aborda questões de projeto de controladores referentes a desempenho a resposta ao degrau.

Verdier *et al.* [75] apresentam um método automático de síntese de controladores para sistemas de amostragem de dados não-lineares com especificações de segurança e alcançabilidade. Basicamente, a estratégia apresentada não se limita aos sistemas polinomiais e controladores. Considera-se ocasionalmente controladores comutados baseados em funções de barreira de controle Lyapunov. A estratégia proposta utiliza programação genética para sintetizar essas funções, bem como os modos do controlador. A exatidão do controlador é verificada através de um solucionador de hipótese de módulo de satisfiabilidade. De acordo com o estudo apresentado, a metodologia utilizada apresentou bons resultados, considerando o que os autores se propuseram a fazer. Apesar disso, testaram apenas em sistemas até terceira ordem. Além do mais, trabalharam somente com sistemas contínuos considerando somente estabilidade do sistemas e desconsiderando aspectos de implementação em sistemas microprocessados, como os efeitos de FWL.

Ter um controlador que atenda a requisitos de projeto é cruciais para seu correto de-

sempenho. A síntese é uma técnica muito poderosa de se ter um controlador que atenda aos requisitos de projetos. O presente trabalho visa o atendimento a requisitos de desempenho (tempo de assentamento e sobressinal) e de implementação (efeitos de FWL nos controladores). A técnica de síntese utilizada neste trabalho é baseada do método de CEGIS, onde uma de suas etapas é a verificação desses requisitos. Daí a importância da integração das técnicas de verificação na *engine* de síntese utilizada nesta pesquisa. Outra etapa no processo de síntese é a geração de controladores candidatos, para isso, utilizou-se algoritmo genético, visto que é uma técnica bastante disseminada em diferentes aplicações. Dentro os trabalhos considerados, nenhum deles se propôs a enfrentar todos os problemas que a presente pesquisa busca resolver, tais como a consideração de requisitos de desempenho em sistemas de controle de tempo discreto e os efeitos de FWL nos controladores empregados nesses sistemas.

3.3 Comparação entre os Trabalhos

Os trabalhos relacionados tratam de verificação de sistemas de controle que é uma das fases deste trabalho, e síntese de controladores de sistemas de controle. Pode-se notar a respeito dos trabalhos analisados que o problema de síntese de controladores digitais é pertinente, visto que este processo é bastante necessário na fase de projeto de um controlador digital. A síntese de controladores digitais para sistemas de controle foi observado como um problema complexo e de difícil implementação, dado que muitos trabalhos abordam o problema de diferentes maneiras e levando em consideração fatores diferentes. Sabendo disso, as principais diferenças entre a abordagem proposta nesse trabalho para as aqui discutidas podem ser listadas: trabalha com sistemas de controle em tempo discreto o qual é o que efetivamente é usado quando se implementa numa plataforma microprocessada. A metodologia proposta sintetiza e verifica controladores considerando requisitos de desempenho de sistemas de controle (tempo de assentamento e sobressinal). Por último, consideram-se aspectos de implementação que podem ocasionar mau funcionamento dos sistemas de controle, visto que os efeitos de FWL, por exemplo, podem levar a isso. A Tabela 3.1 mostra um quadro comparativo entre a metodologia proposta nesta dissertação e os trabalhos relacionados selecionados.

O suporte a requisitos de desempenho descrito na Tabela 3.1, refere-se, especificamente a tempo de assentamento e sobressinal. Dentre os trabalhos catalogados e analisados, pode-se perceber que 20%, abordam o esquema de síntese com verificação de controladores, onde ne-

Tabela 3.1: Comparação dos trabalhos relacionados

Trabalhos relacionados	**Suporte a sistemas discretos**	**Suporte a requisitos de desempenho**	**Considera os efeitos de FWL**	**Verificação**	**Síntese**
Bessa et al. (2017)	X		X	X	
Wang et al. (2016)	X			X	
Yordanov et al. (2012)	X			X	X
Sadraddini e Belta (2017)	X			X	
Nilsosn et al. (2016)	X		X		X
Gross et al. (2017)		X		X	
Ravanbakhsh e Sankaranarayanan (2016)				X	X
Hassani e Lee (2016)		X		X	
Verdier et al. (2018)					X
Abate et al. (2017)	X		X	X	X
Metodologia proposta	X	X	X	X	X

nhum deles considerou requisitos de desempenho da resposta ao degrau de sistemas de controle.

Outra lacuna observada que chama bastante atenção é a respeito da análise de aspectos de implementação de controladores, mais especificamente os efeitos de FWL, 30% dos trabalhos trataram sobre esse tema.

Dentre os trabalhos analisados, 50% implementam a síntese de controladores de sistemas de controle, onde 60% destes trabalham com sistemas de controle em tempo discreto. Através da Tabela 3.1 pode-se observar que existem trabalhos que desenvolveram somente uma *engine* de verificação, mas não implementaram um esquema de síntese de controladores. Outros trabalhos, criaram esquemas de síntese que não utilizam nenhum tipo de verificação formal para checar as propriedades em questão. Vale ressaltar que nenhum dos trabalhos se preocuparam em criar um esquema de síntese de controladores que busque integrar em seu modelo análises de requisitos de desempenho e de implementação de sistemas de controle em tempo discreto. Com base nesses dados, notou-se essa lacuna e consequentemente resolveu-se desenvolver, primeiramente, uma metodologia de verificação formal de sistemas de controle em tempo discreto, considerando requisitos de desempenho e implementação, e, posteriormente, integrando este resultado a um esquema de síntese para geração de controlador que atenda a esses requisitos.

3.4 Resumo

Neste capítulo foram apresentados diversos trabalhos relacionados a algum dos seguintes problemas: verificação de controladores de sistemas de controle, suporte a sistemas de con-

trole em tempo discreto, síntese de controladores digitais, considerando requisitos de desempenho e efeitos FWL. Após a apresentação de cada técnica, assim como uma breve discussão sobre as vantagens e desvantagens de cada uma delas, foi feita uma comparação com a metodologia proposta neste trabalho. Foi possível observar que a metodologia proposta aborda todos os itens analisados, o que o diferencia dos demais trabalhos. O próximo capítulo apresenta a metodologia proposta para síntese de controladores digitais.

Capítulo 4

A Metodologia Proposta

Neste capítulo, serão descritas as metodologias de verificação de tempo de assentamento e sobressinal propostas, bem como as metodologias de síntese de controladores que atendam a estas especificações. Primeiramente, serão descritas as metodologias de verificação de requisitos de desempenho formalizadas e desenvolvidas neste trabalho. Serão detalhadas as técnicas desenvolvidas para estimar o máximo sobressinal e a invariante utilizada para verificar o tempo de assentamento. Também serão delineados os algoritmos criados para verificação de máximo sobressinal e tempo de assentamento. Por último, será mostrado os esquemas de síntese de controladores desenvolvidos que atendam a requisitos de desempenho (tempo de assentamento e sobressinal) em sistemas de controle, mostrando uma visão geral dos métodos utilizados que se baseiam na técnica de CEGIS, onde será detalhado como os controladores candidatos são gerados, a partir de um algoritmo genético, através da elaboração de um problema de otimização.

4.1 Verificação de Especificação de Desempenho Não-frágil

Neste trabalho utilizou-se uma variação da síntese de CEGIS para sintetizar um controlador digital, de modo a satisfazer os requisitos de máximo sobressinal e de máximo tempo de assentamento. Na estratégia do CEGIS, tem-se basicamente duas etapas principais: síntese e verificação. Nesta seção, mostra-se o método de verificação do tempo de assentamento e do máximo sobressinal dos sistemas de controle digital.

4.1.1 Estimativa do Máximo Sobressinal

Para se verificar o máximo sobressinal em um sistema de controle, necessita-se de um parâmetro formal para checar a satisfabilidade desta propriedade. Para isso, foi desenvolvido o Algoritmo 1 que descreve um procedimento para estimar y_p (valor máximo de pico na resposta de um sistema) e k_p (amostra onde y_p está localizado), a fim de encontrar uma abordagem formal para verificar o máximo sobressinal usado nesta abordagem de CEGIS. Quando se procura o maior pico em um sinal amortecido, o primeiro valor da amostra não necessariamente é o que se busca, pois existem muitos casos onde há ondulações, após o primeiro pico, que possuem picos que são maiores que o primeiro, como ilustrado na Figura 4.1. Para evitar esses falsos positivos, leva-se em consideração os picos locais de um sinal e procura pelo maior deles, em relação ao valor em regime estacionário, seguindo uma lógica de detecção de picos através de gradientes, como mostra o Algoritmo 1.

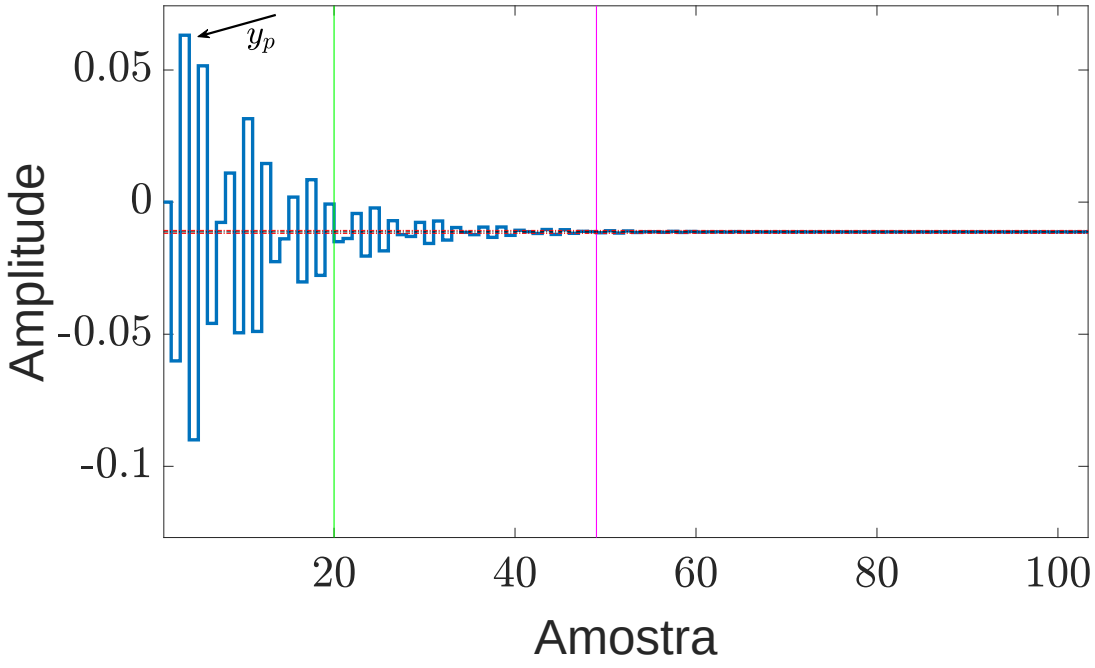


Figura 4.1: Exemplo onde o pico máximo não está na primeira amostra.

Algoritmo 1: `estimate_maxPeak_value()` - Estimação de y_p e k_p

Entrada: **A**, **B**, **C**, **D**, **K** e u
Saída: y_p e k_p
$y_p \leftarrow y(k)$;
$\nabla_{k-1} \leftarrow 1$;
enquanto *verdadeiro* **faça**
  se $|y(k+1)| \geq |y(k)|$ **então**
    $\nabla_k = (\nabla_k > 0)?(\nabla_k + 1) : 1$;
    se $|y(k+1)| != |y(k)|$ **então**
      $y_{\nabla_i} = y(k+1)$;
      $i = k+1$;
    fim
  fim
  caso contrário
    $\nabla_k = (\nabla_k < 0)?(\nabla_k - 1) : -1$;
  fim
  se $(\nabla_{k-1} > 0)$ *e* $(\nabla_k < 0)$ **então**
    se $|y_{\nabla_i}| \leq |y_p|$ **então**
      incrementar ξ;
      se $\xi > 2$ **então**
        interrompa;
      fim
    fim
    caso contrário
      $y_p = y_{\nabla_i}$;
      $k_p = i$;
    fim
  fim
  senão se $(\nabla_k > 10)$ *e* $(|(y(k+1) - y_{ss})/y_{ss}| < 0.001)$ **então**
    se $|y_{ss}| > |y_p|$ **então**
      $y_p = y_{ss}$;
      $k_p = 0$;
    fim
    interrompa;
  fim
  $\nabla_{k-1} = \nabla_k$;
  incrementar k;
fim

No Algoritmo 1, as variáveis ∇_k, y_{∇_i}, i, ξ, e k são referentes, respectivamente, ao gradiente atual, valor em amplitude da amostra onde o primeiro gradiente foi detectado, número da amostra onde essa amplitude está localizada, número de picos detectados que não interessam para a análise (por exemplo, picos detectados que são menores que o atual), e número da amostra atualmente analisada. Como entrada do algoritmo, tem-se as matrizes de estado **A**, **B**, **C** e

D, o controlador **K** (se a análise for em malha fechada) e a entrada do sistemas u. Esses dados de entrada são utilizados para cálculos intermediários, como a obtenção de $y(k)$ e y_{ss}, através de suas respectivas fórmulas (equação 2.10 e equação 2.11). Como saída, o algoritmo produz o pico máximo (y_p) e o instante onde essa amostra está localizada (k_p).

Primeiramente, o primeiro bloco condicional, verifica se o valor da amostra está sempre aumentando, em relação a anterior, para que, então, se possa identificar se há mudança no sentido do gradiente, neste caso, um aumento deste. Caso o valor da próxima amostra seja diferente da atual, esse valor é guardado, como o valor em amplitude da amostra onde o primeiro gradiente foi detectado. Por outro lado, se não houver um aumento no valor da amostra, o gradiente é diminuído (o que não interessa para a análise de detecção dos maiores picos). O segundo bloco condicional verifica se houve uma inversão do sentido do gradiente e tenta descartar aqueles que não interessam, devido a, por exemplo, ser menores que o atual. Caso haja a inversão, é verificado se o valor da amostra onde foi detectado o primeiro pico (y_{∇_i}) é menor ou igual ao maior pico atual, em módulo. Se for maior, trata-se de um pico que não interessa para a análise, caso contrário, o maior pico atual, é o valor, em amplitude, da amostra onde foi detectado o primeiro pico. Em contrapartida, se o valor do gradiente atingir um valor máximo pré-estabelecido (após testes, verificou-se que um gradiente com o valor igual a 10 é suficiente) e a diferença percentual entre o valor da amostra posterior e o valor em regime estacionário for menor que $0,1\%$ (essa diferença foi adotada porque observou-se que é uma quantidade muito pequena), é feito uma verificação se esse valor em regime estacionário é maior que o valor do maior pico atual, em módulo, porque se esse for o caso, adota-se que o y_{ss} é o maior pico (y_p).

4.1.2 Estimação da Invariante de Máximo Tempo de Assentamento

Nesta etapa deste trabalho, deseja-se verificar se um determinado sistema de controle digital atende a um tempo de assentamento requerido, e, como consequência, precisa-se modelar seu comportamento. Tendo isso em vista, foi criada a variável $\hat{k}$ que é uma invariante que auxilia na verificação de tempo de assentamento. Para clarificar o conceito de invariantes pode-se fazer a seguinte análise. Dado um mapeamento ϕ de uma dada coleção M de objetos matemáticos dotados de uma relação de equivalência fixa ρ, em outra coleção N de objetos matemáticos, que é constante nas classes de equivalência de M, com relação a ρ (mais precisamente, isso é uma invariante da relação de equivalência ρ em M). Se X é um objeto em M, então pode-se dizer

que $\phi(M)$ é uma invariante do objeto X [76]. Em outras palavras, uma vez que se calcula $\hat{k}$, ele permanecerá inalterado ao longo do processo de análise do sistema.

Nesse sentido, dado que $S_1 = \{|\lambda_1, |\lambda_2, \ldots, |\lambda_n|\}$ seja o conjunto de valores absolutos de autovalores [58] do sistema Ω e $\overline{\lambda} = \max(S_1)$. Pode-se agora definir uma função heurística que representa a resposta de saída do sistema, $\bar{\Omega}$ como:

$$\bar{\Omega} : \overline{y}(k) = y_{\text{ss}} + \overline{c}\overline{\lambda}^{k}, \tag{4.1}$$

onde $\overline{c}$ é uma constante que faz $\overline{y}(k)$ entrar na região de assentamento Π. Na verdade, a função heurística acima usa o autovalor mais "lento" $\bar{\lambda}$, ou seja, aquele que possui o maior valor absoluto e que é o autovalor dominante, garantindo que $\bar{\Omega}$ sempre alcance a região de assentamento após Ω.

Em resumo, pode-se encontrar uma função heurística baseada no maior autovalor de Ω, em termos de magnitude, e então verificar o momento em que ela entra em uma região de assentamento porque se isso acontecer antes do tempo de assentamento requerido, isso também é válido para Ω.

Para verificar o tempo de assentamento de um sistema, indica-se uma região de assentamento como uma porcentagem p, geralmente entre 0% e 5% [77], portanto, seus limites superior e inferior são,

$$L_{\text{upp}} = \left(1 + \frac{p}{100}\right) y_{\text{ss}} \tag{4.2}$$

e

$$L_{\text{low}} = \left(1 - \frac{p}{100}\right) y_{\text{ss}}, \tag{4.3}$$

respectivamente. O instante ($\hat{k}$) quando um sistema entra em sua região de assentamento pode ser encontrado simplesmente fazendo com que equação (4.1) seja igual a equação (4.2), devido à forma da curva na equação (4.1), como segue:

$$\begin{aligned}
\left(1 + \frac{p}{100}\right) y_{\text{ss}} &= y_{\text{ss}} + \overline{c}\overline{\lambda}^{\hat{k}} \\
y_{\text{ss}} + \frac{p}{100} y_{\text{ss}} &= y_{\text{ss}} + \overline{c}\overline{\lambda}^{\hat{k}} \\
\overline{c}\overline{\lambda}^{\hat{k}} &= \frac{p}{100} y_{\text{ss}} \\
\log_{\overline{\lambda}} \overline{\lambda}^{\hat{k}} &= \log_{\overline{\lambda}} \left(\frac{p}{100\overline{c}} y_{\text{ss}}\right)
\end{aligned}$$

$$\hat{k} = \lceil \log_{\bar{\lambda}} \left(\frac{p}{100\bar{c}} y_{\text{ss}} \right) \rceil. \tag{4.4}$$

O maior pico da saída do sistema, y_{p}, pode ser obtido para calcular $\bar{c}$ e, como consequência, com equação (4.1), faz-se:

$$\begin{aligned} y_{\text{p}} &= y_{\text{ss}} + \bar{c}\bar{\lambda}^{k_{\text{p}}} \\ \bar{c} &= \frac{y_{\text{p}} - y_{\text{ss}}}{\bar{\lambda}^{k_{\text{p}}}}. \end{aligned} \tag{4.5}$$

O valor de y_{p} pode ser obtido, verificando a saída do sistema para encontrar o maior valor em magnitude com o mesmo sinal de y_{ss} e k_{p} é a amostra em que y_{p} está localizado, como elucidado no Algoritmo 1. Tendo calculado o valor de $\bar{c}$, pode-se utilizar a função descrita na equação (4.1) a qual garante que sempre entrará na região de assentamento depois da resposta do sistema em análise. Com isso, viabiliza uma forma de saber se um determinado tempo de assentamento está dentro de um valor máximo que pode ser medido através do valor de $\hat{k}$. Uma explicação mais detalhada de como funciona será mostrada nas seções subsequentes.

4.1.3 Algoritmo de Verificação de Sobressinal

Nesta seção, explica-se nossa técnica de verificação de máximo sobressinal, que consiste em verificar se o percentual exigido de sobressinal é menor ou igual ao percentual de sobressinal real (calculado), de modo a se identificar uma violação.

O máximo sobressinal (M_{p}) é o valor máximo transitório que excede y_{ss}. A fim de encontrar a porcentagem de sobressinal, executa-se a seguinte computação:

$$\begin{aligned} M_{\text{p}} &= y_{\text{p}} - y_{\text{ss}} \\ PO &= 100 \times \frac{M_{\text{p}}}{y_{\text{ss}}}, \end{aligned} \tag{4.6}$$

onde y_{p} pode ser obtido do Algoritmo 1.

A Figura 4.2 mostra a metodologia de verificação de sobressinal proposta, que foi implementada no DSVerifier. Em primeiro lugar, o modelo de um sistema de controle (`Step 1`) é determinado e, em seguida, um controlador é projetado em malha fechada (`Step 2`). Depois disso, a implementação de FWL (`Step 3`) e o máximo sobressinal requerido para ser verificado (`Step 4`) são definidos, os quais são então usados para criar o arquivo `SpecsFile.ss`.

Em seguida, a estabilidade do sistema é verificada e, se for estável, a especificação do máximo sobressinal é avaliada.

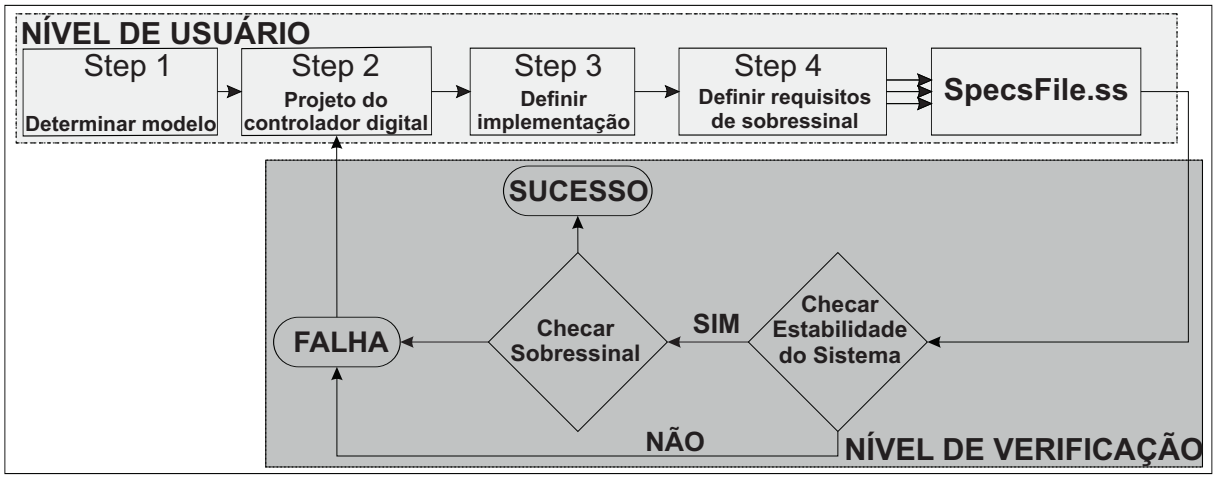


Figura 4.2: Verificação de Sobressinal.

A verificação de máximo sobressinal consiste basicamente em se avaliar se a porcentagem de sobressinal computada $P.O.$ (equação (4.6)) é menor que o percentual de sobressinal requerido ($P.O._{\mathrm{r}}$). Se for esse o caso, ele retorna "`Verification SUCCESSFUL`"; caso contrário, "`Verification FAILED`" será exibido, conforme ilustrado no Algoritmo 2.

Algoritmo 2: Verificação de sobressinal.

Dados: y_{ss}, PO_{r}

$y_{\mathrm{p}} \leftarrow estimate\_maxPeak\_value()$;

$PO \leftarrow \frac{y_{\mathrm{p}} - y_{\mathrm{ss}}}{y_{\mathrm{ss}}}$;

se $PO > PO_{\mathrm{r}}$ **então**

    Verification Failed;

senão

    Verification Successful;

fim

4.1.4 Algoritmo de Verificação de Tempo de Assentamento

Nesta seção, mostra-se o algoritmo para verificação de tempo de assentamento, onde se usa o invariante $\hat{k}$ (cf. Seção 4.1.1). Nossa metodologia, que é ilustrada na Figura 4.3, fornece verificação de tempo de assentamento em sistemas em malha aberta e fechada; no entanto, os primeiros não apresentam controladores e, consequentemente, os efeitos de FWL não são con-

siderados. A primeira etapa consiste em obter os parâmetros de entrada necessários, matrizes de espaço de estados **A**, **B**, **C** e **D** e uma entrada do sistema **u**, que é seguida por um controlador digital, se um sistema em malha fechada for considerado (`Step 2`). Então, o tempo de assentamento requerido t_{sr}, a porcentagem p da região de assentamento e o tempo de amostragem T_s são definidos (`Step 4`), que é seguido por uma implementação de FWL, quando em malha fechada (`Step 3`). Depois disso, os passos `A` a `E` são executados, *i.e.*, cálculo do controlador com FWL, se for escolhido em malha fechada (`Step A`) e valor de estado estacionário (y_{ss}), como descrito em equação (2.11) (`Step B`), estimativa dos maiores valores de pico da saída do sistema (y_p) e a amostra k_p onde y_p está localizado (`Step C`) e, finalmente, computação de $\overline{\lambda}$ (`Step D`), $\overline{c}$ e $\hat{k}$ (`Step E`).

Algoritmo 3: Procedimento para encontrar o instante k_r

Enquanto $y(k) \not\subset \Pi$ **faça**

  incrementar k;

fim

$k_r \leftarrow k$;

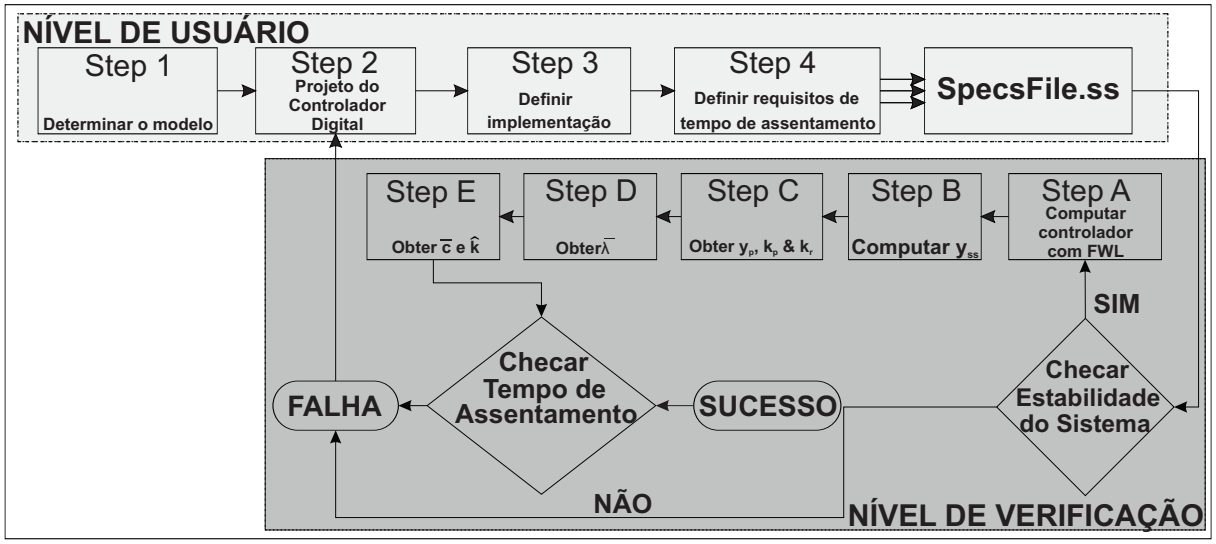


Figura 4.3: Metodologia de Verificação de Tempo de Assentamento.

O procedimento proposto para verificar os tempos de assentamento, que é descrito no Algoritmo 4, consiste em primeiro verificar se $\hat{k}$, que foi previamente calculado e usado como parâmetro, é menor que tempo de assentamento requerido $k_{sr} = \lceil \frac{t_{sr}}{T_s} \rceil$, que também pode ser verificado através de $\hat{k} \times T_s \leq t_{sr}$ (cf. Algoritmo 4). Se este for o caso, pode-se assegurar que o sinal de saída está dentro de Π (cf. Seção 4.1.1) após t_{sr} e, como consequência, a verificação

associada já é bem-sucedida; caso contrário, ainda é preciso verificar se y permanece dentro de Π, entre k_{sr} e $\hat{k}$, e, se este último não for verdadeiro para qualquer amostra y, a verificação resultante falha. Dado que se utiliza o autovalor mais "lento"como base da exponencial descrita na equação (4.1), notou-se através dos experimentos com os *benchmarks* que a função convergia de forma atender o que se procurava, encontrar uma função que sempre teria um comportamento mais lento que a reposta do sistema original.

Algoritmo 4: Verificação de tempo de assentamento.

```
se y_p ⊂ Π então
    k̂ ← k_r;
senão
    k ← t_sr / T_s;
    se k > k̂ então
        k̂ ← k_r;
    senão
        Enquanto k ≤ k̂ faça
            se y(k) < L_low  ou  y(k) > L_upp então
                Verification Failed;
            fim
            incrementar k;
        fim
    fim
fim
se t_sr < k̂ × T_s então
    Verification Failed;
senão
    Verification Successful;
fim
```

4.1.5 Exemplo de Verificação

A fim de fornecer uma compreensão mais profunda sobre os algoritmos de verificação de máximo sobressinal e tempo de assentamento, a seguir é apresentado um exemplo prático. Começando com a metodologia proposta para a verificação de máximo sobressinal, tem-se que definir um sistema e depois analisá-lo, de acordo com a nossa abordagem:

$$\Omega : \begin{cases} \mathbf{x}(k+1) = \begin{bmatrix} 1.5 & 1.0 & 0.0 \\ 0.0 & 1.5 & 1.0 \\ 0.0 & 0.0 & 1.5 \end{bmatrix} \mathbf{x}(k) + \begin{bmatrix} -0.4 \\ 2.5 \\ -0.8 \end{bmatrix} \mathbf{u}(k) \\ \mathbf{y}(k) = \begin{bmatrix} 0.0 & 2.6 & 0.0 \end{bmatrix} \mathbf{x}(k) + [0]\mathbf{u}(k) \\ \mathbf{u}(k) = \mathbf{r}(k) - \mathbf{K}\mathbf{x}(k) \end{cases}, \tag{4.7}$$

onde $\mathbf{r}(k) = u_{-1}$ é o degrau unitário.

Desta forma, `Step 1` está pronto e `Step 2` é então considerado, onde uma matriz do controlador $\mathbf{K}$ não está definida, devido à operação em malha aberta. Consequentemente, uma implementação de FWL (`Step 3`) também não é fornecida. Em seguida, os requisitos desejados são definidos, como $PO_{\mathrm{r}} = 9$, em `Step 4`, usando um arquivo de especificação `SpecsFile.ss`.

Dado que a operação em malha aberta foi considerada, este sistema é avaliado com "`dsverifier SpecsFile.ss --property OVERSHOOT`" e, como consequência, "`Verification FAILED`" é retornado, porque o sistema avaliado é instável e, consequentemente, não há sobressinal. Pode-se notar que o método proposto sempre verifica se um sistema é estável, o que é condição para a verificação falhar.

Como o sistema escolhido é instável, pode-se voltar para `Step 2` e projetar um controlador, de modo a estabilizá-lo e satisfazer o percentual de sobressinal requerido (PO_{r}), usando a técnica de Lyapunov [58]. Nesse caso, usa-se um controlador $\mathbf{K} = [-0.7351 \ -5.0045 \ -18.5371]$ em `SpecsFile.ss`. Em seguida, executa-se esse experimento novamente, mas agora com a opção "`--closed-loop`" e ainda sem efeitos FWL (`--no-fwl`), o que resulta em "`Verification SUCCESSFUL`", desde que o DSVerifier retornou $PO = 2.6228$ que é menor que $PO_{\mathrm{r}} = 9$.

Finalmente, consideram-se os efeitos de FWL nesse experimento. Isso é feito usando o formato $\langle 4,4 \rangle$, a matriz do controlador $\mathbf{K}_{\mathrm{FWL}} = [-0.6875 \ -5.0 \ -18.5]$ e omitindo "`--no-fwl`" (`Step A`), que resulta em "`Verification FAILED`", devido ao fato de o sistema ainda estar instável. Para avaliar outros formatos, $\langle 8,8 \rangle$ e $\langle 16,16 \rangle$ foram considerados, em `Step 2`. Como consequência, $\langle 8,8 \rangle$ ($\mathbf{K}_{\mathrm{FWL}} = [-0.7344 \ -5,0039 \ -18.5352]$) resultou em "`Verification SUCCESSFUL`", dado que o DSVerifier retornou $PO = 1.6153$, que é menor que $PO_{\mathrm{r}} = 9$ e $\langle 16,16 \rangle$ ($\mathbf{K}_{\mathrm{FWL}} = [-0.7351 \ -5.0045 \ -18.5370]$) também teve como resultado "`Verification`

SUCCESSFUL", porque a metodologia forneceu $PO = 2.6253$, que é novamente menor que o valor requerido.

Agora, analisa-se a abordagem de verificação de tempo de assentamento proposta, seguindo os passos da Figura 4.3. Desta forma, Step 1 está pronto e Step 2 é então considerado, onde uma matriz do controlador **K** não está definida, devido à operação em malha aberta. Consequentemente, uma implementação de FWL (Step 3) também não é fornecida. Em seguida, os requisitos desejados são definidos, ou seja, $t_{sr} = 10s$, $p = 5$ e $T_s = 0.5s$, em Step 4, usando um arquivo de especificação SpecsFile.ss.

Finalmente, considerando que a operação em malha aberta foi considerada, este sistema é avaliado com "dsverifier SpecsFile.ss - property SETTLING_TIME". Steps A a E da metodologia proposta são automaticamente executados pelo DSVerifier, considerando uma implementação em malha aberta, e, como consequência, "Verification FAILED" é retornado, porque o sistema avaliado é instável, pois, seus autovalores são maiores que 1 ($\lambda = 1.5$, devido à equação (2.6)). Pode-se notar que o método proposto primeiro verifica se um sistema é estável, entre Steps A e B.

Como o sistema escolhido é instável, pode-se voltar para o Step 2 e projetar um controlador, a fim de estabilizá-lo e satisfazer o tempo de assentamento requerido, usando o Lyapunov [58]. Nesse caso, usa-se a mesma matriz do controlador $\mathbf{K} = [-0.7351 \ -5.0045 \ -18.5371]$ em SpecsFile.ss, como explicado em Seção 4.1.3. Como esse controlador foi projetado para satisfazer ambas as propriedades (tempo de assentamento e sobressinal), esse experimento é executado novamente, mas agora com a opção "--closed-loop" e ainda sem efeitos FWL (--no-fwl), que resulta em "Verification SUCCESSFUL", como pode ser visto na Figura 4.4(a), onde a resposta ao degrau não deixa a região de assentamento entre k_{sr} e $\hat{k}$.

Finalmente, esse experimento pode considerar os efeitos de FWL. Isso é feito usando o formato $\langle 4,4 \rangle$, a matriz do controlador $\mathbf{K}_{FWL} = [-0.6875 \ -5.0 \ -18.5]$ e omitindo --no-fwl (Step A), que resulta em "Verification FAILED", como mostrado na Figura 4.4(b), onde o sistema se torna instável. Vale a pena notar que a matriz do controlador **K** em SpecsFile.ss não é modificada e $\mathbf{K}_{FWL}$ é internamente calculado pelo DSVerifier.

Para avaliar outros formatos, $\langle 8,8 \rangle$ e $\langle 16,16 \rangle$ são considerados em Step 2. Como consequência, $\mathbf{K}_{FWL} = [-0.7344 \ -5.0039 \ -18.5352]$ com "Verification FAILED", porque a resposta ao degrau deixou a região de assentamento entre k_{sr} e $\hat{k}$ e $\mathbf{K}_{FWL} = [-0.7351 \ -5.0045 \ -18.5370]$ com "Verification SUCCESSFUL", porque a resposta ao degrau não dei-

xou a região de assentamento entre k_{sr} e $\hat{k}$, são respectivamente obtidas, como pode ser visto nas Figuras 4.4 (c) e (d).

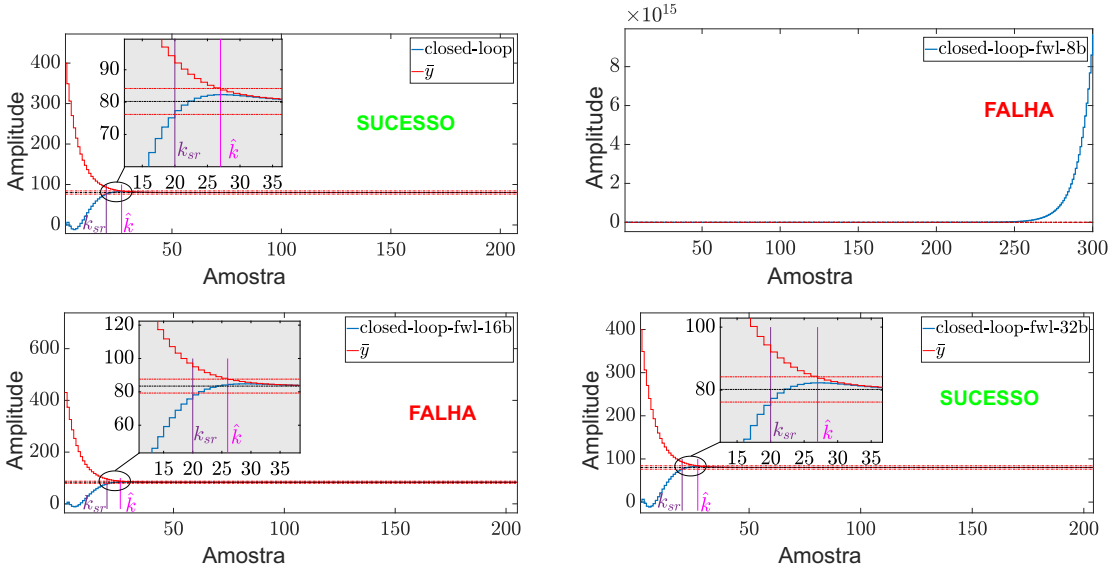


Figura 4.4: Verificação de tempo de assentamento para o sistema do exemplo, (a) sem e com os efeitos de FWL nos formatos (b) $\langle 4,4\rangle$, (c) $\langle 8,8\rangle$, e (d) $\langle 16,16\rangle$.

O número de *bits* influencia fortemente os tempos de assentamento em implementações reais, devido a efeitos de FWL. Além disso, "`Verification FAILED`" foi obtido para um formato intermediário $\langle 8,8\rangle$, enquanto foi bem-sucedido com $\langle 16,16\rangle$. De fato, dado que o tempo de assentamento resultante é principalmente dependente da resposta natural mais rápida e sua taxa de amortecimento, a interação entre eles após a quantificação pode causar tal comportamento, o que reforça ainda mais o uso da metodologia proposta, durante fases de projeto.

Pode-se notar também que as propriedades abordadas durante as fases de projeto também são modificadas, devido a mudanças nos coeficientes e nos resultados de cálculo. Em particular, o sobressinal foi afetado de maneiras diferentes, o que significa que, se uma propriedade falhar, a outra talvez não falhe, como pode ser visto no formato $\langle 8,8\rangle$.

Pode-se notar que $t_{\text{sr}} = 10s$ é a amostra $n = 20$, discretizada em $T_{\text{s}} = 0.5s$. A Figura 4.4 (a) mostra que $k_{\text{sr}} = \lceil t_{\text{sr}}/T_{\text{s}} \rceil = 20$ é menor que o $\hat{k} = 37$ calculado (equação (4.4)) e, como consequência, para este tempo de assentamento requerido, o sistema já está na região de assentamento, para que o tempo de assentamento seja possível nesse caso. Finalmente, o mesmo raciocínio é válido para as Figuras 4.4 (b), (c) e (d), onde se pode analisar as respostas e verificá-las de acordo com o Algoritmo 4.

As metodologias de verificação de máximo tempo de assentamento e de sobressinal, por

si só, já criariam uma estratégia iterativa de projeto e de verificação, que convergiria para uma implementação mais adequada. Além disso, podem ser colocadas como metodologias passíveis de implementação em linguagens de programação, bem como, integradas a um esquema automático que é o objetivo final desta pesquisa.

4.2 Síntese de Requisitos de Especificações de Desempenho Não Frágil

Nesta seção, mostra-se a metodologia de síntese proposta com base nos requisitos de especificação de desempenho para sistemas de controle digital. Em particular, analisam-se as especificações de tempo de assentamento e máximo sobressinal, para sistemas de controle com realimentação de estado digital. A abordagem de síntese proposta, que emprega nossas metodologias de verificação para o máximo tempo de assentamento e máximo sobressinal, como explicado nas seções 4.1.3 e 4.1.4, é baseada na técnica de síntese CEGIS, onde existem dois estágios principais: verificação e aprendizado (onde o sintetizador está localizado) como pode-se ver no diagrama da Figura 2.5. No estágio de aprendizado é gerado um controlador candidato **K** a fim de atender a especificação requerida (tempo de assentamento e/ou sobressinal), usando algoritmo genético [25], pois trata-se de uma técnica bastante difundida para este fim. Depois de gerar o controlador **K**, há o estágio de verificação onde ele pega o controlador **K** e verifica se atende aos requisitos, de acordo com nossas metodologias de verificação, e se falhar nessa etapa, ele volta ao estágio de aprendizado e se mantém nesse ciclo, até obter sucesso na verificação ou o sintetizador atingir um número máximo de tentativas.

A Figura 4.5 mostra a arquitetura de síntese proposta de uma forma geral. O bloco de definições do sistema representa todas as característica intrínsecas do sistema (matrizes de espaço de estados, entradas, etc.) e os requisitos a serem sintetizados (tempo de assentamento e sobressinal). A variável `total_tentativas` é o número total de tentativas de sintetizar um controlador. Já a variável `tentativas` representa o número de tentativas sem alterar qualquer parâmetro de síntese (tamanho da população, número de gerações do algoritmo genético, etc). O bloco GA realiza a geração de um controlador candidato através de algoritmo genético. Já o bloco `verify()` é o mecanismo de verificação (Seções 4.1.3 e 4.1.4) que depende do que se queira sintetizar (tempo de assentamento e/ou sobressinal). O parâmetro `MAXTENT` é o nú-

mero máximo de tentativas para se tentar sintetizar um controlador (informado pelo usuário). `MAXTENT_INT` representa o número máximo de tentativas sem alterar nenhum parâmetro do GA. O parâmetro `popsize` é o tamanho da população do algoritmo genético e `STEP` é o passo para incrementar a população GA na *engine* de síntese desenvolvida neste trabalho.

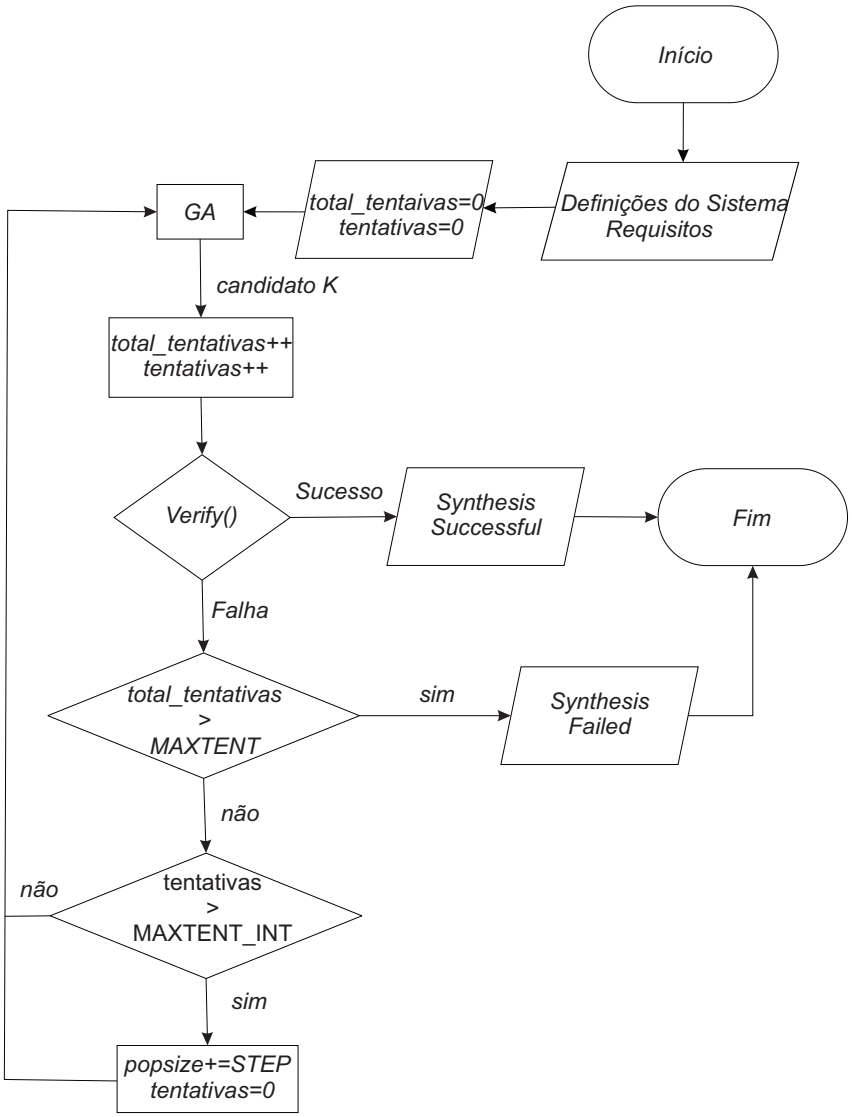


Figura 4.5: Arquitetura da metodologia de síntese proposta.

A metodologia de síntese proposta funciona da seguinte maneira. Primeiro define-se o sistema a ser sintetizado (planta, entradas, etc) e os requisitos a serem atendidos (tempo de assentamento/sobressinal), onde, neste trabalho, todas essas informações são colocadas em um arquivo de configuração. Além disso, executa-se o algoritmo genético sob as restrições necessárias (tempo de assentação e/ou sobressinal). Em seguida, o GA gera um controlador candidato **K** para realizar o processo de verificação deste, aplicado ao sistema sob análise. Se a veri-

ficação for bem-sucedida, a síntese é finalizada com sucesso. Caso contrário, a verificação falha, ou seja, não conseguiu encontrar a melhor solução, e verifica, então, se o número total de tentativas (`total_tentativas`) excedeu o número máximo de tentativas (`MAXTENT`), se sim, o processo de síntese é malsucedido e termina, senão, verifica-se se o número de tentativas (`tentativas`), sem a mudança de qualquer parâmetro do GA, excedeu o número máximo definido(`MAXTENT_INT`), e caso tenha excedido, volta a executar o GA novamente. Caso contrário, incrementa-se o tamanho da população do GA, por um passo fixo (`STEP`), e depois disso executa novamente o GA e o processo continua. Todas essas informações de entrada são retidas dependendo de como a metodologia de síntese é implementada, em uma linguagem de programação. Neste trabalho, essas informações de entrada são informadas em um arquivo de configuração, sendo que outras informações referentes a escolha de quais especificações de desempenho deseja-se sintetizar, são informadas por linha de comando, como será mostrada mais adiante.

4.2.1 Geração do Controlador Candidato K

Nesta metodologia de síntese baseada em CEGIS, utiliza-se algoritmos genéticos para gerar um controlador candidato **K**, a fim de satisfazer os requisitos desejados (tempo de assentamento e sobressinal). Deseja-se elaborar um problema de otimização multiobjetivo, onde se definiu duas funções de custo para resolver três problemas de otimização separadamente, considerante sobressinal, tempo de assentamento ou ambos. As funções de custo são:

$$\mathbf{f}_1(\mathbf{K}) = \mathbf{K} \times \mathbf{K}^T \tag{4.8}$$

e

$$\mathbf{f}_2(\mathbf{K}) = \hat{k} \tag{4.9}$$

onde **K** é o controlador e $\hat{k}$ é dado pela equação (4.4). A função $\mathbf{f}_1$ é a multiplicação do vetor **K**, que representa o controlador, pelo seu transposto. Essa operação é o quadrado do módulo do vetor de ganhos, ou seja, é menor quando menor for o valor absoluto dos ganhos. Baixos ganhos são bons pois produzem controladores menos sensíveis a ruídos e evita saturações [24]. Por outro lado, a função $\mathbf{f}_2$ é a função que representa o valor da invariante $\hat{k}$, e foi escolhida por está diretamente relacionada com o problema (tempo de assentamento). É importante que essas

restrições sejam completamente atendidas para que se consiga gerar um candidato válido para o esquema de síntese.

Problema de Otimização para Sobressinal

Com o objetivo de gerar uma matriz de controlador **K** através de algoritmos genéticos, de modo a satisfazer os requisitos de sobressinal, utiliza-se a seguinte formulação do problema.

O problema de otimização consiste em minimizar $\mathbf{f}_1$ e $\mathbf{f}_2$ em relação à variável de decisão **K**. Assim, o problema de otimização é representado da seguinte forma:

$$\begin{aligned} \min \quad & (\mathbf{f}_1(\mathbf{K}), \mathbf{f}_2(\mathbf{K})), \\ \text{sujeito a} \quad & PO \leq PO_{\mathrm{r}}, \\ & \bar{\lambda} \leq 1 \end{aligned} \tag{4.10}$$

A primeira restrição significa que a porcentagem real de sobressinal deve ser menor que a porcentagem requerida (PO_{r}). A segunda restrição está relacionada à estabilidade do sistema, onde $\bar{\lambda}$ é o valor absoluto máximo entre os autovalores do sistema (cf. Seção 4.1.1) e este valor deve ser menor que 1, caso contrário, o sistema é instável [58]. O mecanismo de síntese somente usa os candidatos gerados pelo GA que atendam às restrições do problema de otimização.

Problema de Otimização para Tempo de Assentamento

Desta vez, deseja-se otimizar o problema utilizando os requisitos de tempo de assentamento. Para isso, utiliza-se a mesma função multi-objetivo a qual se almeja minimizar $\mathbf{f}_1$ e $\mathbf{f}_2$ em relação à variável de decisão **K**. Assim, o problema de otimização é representado da seguinte forma:

$$\begin{aligned} \min \quad & (\mathbf{f}_1(\mathbf{K}), \mathbf{f}_2(\mathbf{K})), \\ \text{sujeito a} \quad & \hat{k} \leq k_{\mathrm{sr}}, \\ & \bar{\lambda} \leq 1 \end{aligned} \tag{4.11}$$

Neste problema de otimização, a primeira restrição significa que $\hat{k}$ deve ser menor que o tempo de assentamento requerido em tempo discreto (k_{sr}), porque como pode ser visto no Algoritmo 4, quando isso acontece, a verificação é bem-sucedida, o que significa que o sistema

atende ao máximo tempo de assentamento exigido. A segunda restrição está relacionada à estabilidade do sistema, da mesma forma como é mostrado no problema na Seção 4.2.1.

Para que o GA possa fornecer um candidato para o mecanismo de síntese, este deve atender a todas as restrições do problema de otimização.

Problema de Otimização para Satisfazer Sobressinal e Tempo de Assentamento

A fim de gerar um controlador $\mathbf{K}$ usando algoritmo genético que satisfaça tanto os requisitos de sobressinal quanto de tempo de assentamento, formula-se o seguinte problema de otimização.

$$\begin{aligned} \min \quad & (\mathbf{f}_1(\mathbf{K}), \mathbf{f}_2(\mathbf{K})), \\ \text{sujeito a} \quad & \hat{k} \leq k_{\text{sr}}, \\ & PO \leq PO_{\text{r}}, \\ & \bar{\lambda} \leq 1 \end{aligned} \tag{4.12}$$

Neste caso, deseja-se minimizar as funções multi-objetivo ($\mathbf{f}_1$ e $\mathbf{f}_2$) restritas a: $\hat{k} \leq k_{\text{sr}}$, $PO \leq PO_{\text{r}}$ e $\bar{\lambda} \leq 1$. A primeira restrição garante que o sistema atenda ao requisito de tempo de assentamento, já que $k_{\text{sr}} \geq \hat{k}$ (cf. Algoritmo 4). A segunda restrição assegura o percentual máximo de sobressinal exigido (cf. Algoritmo 2). Finalmente, a terceira restrição assegura que o controlador $\mathbf{K}$ mantenha o sistema estável, considerando que $\bar{\lambda}$ é o maior valor absoluto entre os autovalores e que este sendo menor que 1 proporciona estabilidade.

Esse problema de otimização possui mais restrições que os dois primeiros e diante disso, apresenta uma maior dificuldade para se gerar um controlador candidato que atenda a todas a restrições.

4.2.2 Implementação e Uso da Metodologia no DSVerifier

A metodologia de síntese apresentada anteriormente foi implementada dentro da ferramenta DSVerifier, com o objetivo de experimentar as técnicas desenvolvidas neste trabalho como uma ferramenta em diversos *benchmarks*. Basicamente seguiu-se a arquitetura mostrada no diagrama da Figura 4.5.

A funcionalidade de síntese foi implementada em C++, onde todas as operações matriciais necessárias foram realizadas com o auxílio da biblioteca Eigen [78], e o algoritmo genético utilizado foi implementado com a ajuda da biblioteca Galgo [79], disponível para C++, onde se utilizou essa biblioteca como ferramenta de otimização para geração de controladores candidatos. Primeiramente implementaram-se os algoritmos de verificação de tempo de assentamento e sobressinal mostrados na Seção 4.1. Em seguida implementou-se a metodologia da arquitetura mostrada na Figura 4.5.

Para utilizar o DSVerifier com o objetivo de realizar síntese de controladores digitais, usa-se o comando como segue:
"`dsverifier SpecsFile.ss --synthesize [--st] [--os] --maxattempts attempts`", onde as opções "`--st`" e "`--os`" são referentes a opção de se sintetizar controladores considerando somente tempo de assentamento e sobressinal, respectivamente. Porém, desejando-se sintetizar controladores considerando ambas as opções, ou coloca-se explicito no comando (`--st --os`) ou ignora ambas as opções (por padrão). Também define-se no comando o número máximo de tentativa de sintetizar o controlador, com a opção "`--maxattempts`", seguido pelo valor máximo de tentativas.

Exemplo

Para ilustrar o uso da metodologia, o exemplo a seguir é mostrado.

Dado o sistema abaixo, tem-se,

$$\Omega : \begin{cases} \mathbf{x}(k+1) = \begin{bmatrix} -0.5 & 0.4 \\ -0.4 & -0.5 \end{bmatrix} \mathbf{x}(k) + \begin{bmatrix} 0.0 \\ 2.5 \end{bmatrix} \mathbf{u}(k) \\ \mathbf{y}(k) = \begin{bmatrix} 0.0 & 2.6 \end{bmatrix} \mathbf{x}(k) + [0]\mathbf{u}(k) \\ \mathbf{u}(k) = \mathbf{r}(k) - \mathbf{K}\mathbf{x}(k) \end{cases}, \tag{4.13}$$

onde $\mathbf{r}(k) = u_{-1}$ é o degrau unitário.

Neste caso, deseja-se sintetizar um controlador que faça o sistema em (4.13) obedecer ao requisito de tempo de assentamento. Agora, define-se o arquivo de especificação (`SpecsFile.ss`) com todas as especificações de síntese que se deseja. As seguintes especificações serão utilizadas no arquivo de especificação:

```
implementation ⟨16,16⟩
range [1,1]
nStates = 2;
nInputs = 1;
nOutputs = 1;
A = [-0.5,0.4;-0.4,-0.5]
B = [0.0;2.5]
C = [0.0,2.6]
D = [0.0]
x0 = [0.0;0.0]
inputs = [1.0]
tsr = 3.0;
POr = 0.0;
ts = 0.5;
p = 5.0;
bounds_l = -0.5;
bounds_u = 0.5;
popsize = 100;
nbgen = 100;
```

Figura 4.6: Especificação do sistema de controle digital em (4.13) para o DSVerifier.

Primeiramente, define-se no arquivo a especificação de desempenho desejada, no caso, tempo de assentamento quando submetido a efeito de FWL com 32 *bits* (`implementation` $\langle 16,16\rangle$) e a faixa de valores nas variáveis operadas (`range` $[1,1]$). Depois define-se o número de estados, entradas e saídas (`nStates`, `nInputs` e `nOutputs`, respectivamente). Em seguida definem-se as matrizes de estados, depois o estado inicial do sistema (**x0**), entrada (degrau unitário), tempo de assentamento (`tsr`), portentagem de sobressinal `POr=0.0` (foi deixado `POr=0.0` mesmo não considerando esse requerimento), tempo de amostragem (`ts`) e porcentagem da região de assentamento (`p`). Em seguida, definem-se os parâmetros do algoritmo genético de margens superior e inferior inicial dos valores (`bounds_u` e `bounds_l`), o tamanho da população do algoritmo genético (`popsize`) e o número de gerações do GA (`nbgen`). Os valores relativos aos parâmetros de algoritmo genético foram escolhidos empiricamente durante experimentos.

Executando o comando "`dsverifier SpecsFile.ss --synthesize --st --maxattempts 200`", gerou-se o controlador sintetizado com sucesso

$K = [-0.093750000003072001, -0.197326660162715994]$ após 7 tentativas.

Implementação da Verificação no DSVerifier

Esta seção visa mostrar que as metodologias de verificação podem ser implementadas e utilizadas separadamente da síntese. Para isso, será ilustrado a implementação no DSVerifier a opção de verificar controladores, já prontos e projetados em outras plataformas, com respeito a parâmetros de desempenho (tempo de assentamento e sobressinal). Desenvolveu-se os comandos da seguinte forma:

"`dsverifier SpecsFile.ss --property [SETTLING_TIME / OVERSHOOT] [--closed-loop] [--no-fwl]`".

Onde é definido se deseja verificar tempo de assentamento ou sobressinal (`SETTLING_TIME` ou `OVERSHOOT`, respectivamente). Caso se deseje verificar em malha fechada, basta usar a opção "`--closed-loop`", em malha aberta, basta ignorar a opção. Também define-se se deseja considerar os efeitos FWL ou não na verificação, se não, basta considerar a opção "`--no-fwl`".

Define-se no arquivo de especificação praticamente igual como se fez para síntese, porém, não definindo os parâmetros relativos a síntese (`bounds_u`, `bounds_l`, `popsize` e `nbgen`).

A ferramenta desenvolvida retorna "`Verification SUCCESSFUL`", se o sistema de controle informado atende a especificação escolhida e "`Verification FAILED`" caso contrário. Um exemplo prático com o uso do DSVerifier como verificador das especificações de desempenho (tempo de assentamento e sobressinal) é mostrado na Seção 4.1.5.

4.3 Resumo

Neste capítulo foi apresentado a metodologia proposta para sintetizar controladores digitais satisfazendo restrições de especificação de desempenho da resposta ao degrau, considerando os efeitos de FWL, sendo esta a contribuição maior do presente trabalho. Foi dada uma explicação detalhada de todo o processo de síntese a qual foi baseada na técnica de CEGIS. A técnica de síntese empregada neste trabalha possui uma etapa de verificação, onde se mostrou a metodologia que foi criada para fazer a devida verificação de especificações de desemprenho (máximo sobressinal e máximo tempo de assentamento). Mostrou-se também um exemplo prático do funcionamento dos verificadores aplicado ao sistema de controle do mundo real. Por fim, mostrou-se como foi idealizada a etapa de síntese, onde o problema foi modelado como um problema de otimização e utilizou-se algoritmo genético para resolver. Como resultado, tem-se

o embasamento para a avaliação experimental realizada no próximo capítulo.

Capítulo 5

Resultados e Discussões

Este capítulo está dividido em quatro partes. A primeira parte descreve os objetivos dos experimentos conduzidos neste trabalho. A segunda é dedicada à descrição do ambiente computacional o qual os experimentos foram submetidos, máquinas (processador e memória RAM) e sistemas operacionais. A terceira mostra uma descrição dos *benchmarks* criados e utilizados para validar a metodologia desta pesquisa. A quarta apresenta os resultados obtidos quando se realizaram os experimentos com os *benchmarks* selecionados, assim como uma discussão sobre os resultados obtidos e avaliação geral do método proposto nesta dissertação.

5.1 Objetivos do Experimento

Para validar a metodologia desenvolvida neste trabalho de forma mais efetiva, foram desenvolvidas funcionalidades, implementadas em C++, como parte do DSVerifier. Com a implementação destas, pode-se submeter a metodologia a um conjunto de *benchmarks* que foram criados exclusivamente para este trabalho.

Utilizando os *benchmarks* propostos que serão descritos mais adiante, os experimentos têm os seguintes objetivos:

a) Demonstrar a aplicabilidade da metodologia para a síntese de controladores digitais.

b) Avaliar as vantagens da metodologia comparada a outras existentes.

5.2 Ambiente Computacional Experimental

Os experimentos foram conduzidos em diferentes máquinas com configurações distintas, para acelerar a geração dos resultados, dado que eles demandam tempo considerável para sua devida finalização, pois consomem bastante processamento da máquina, e caso fossem executados todos na mesma máquina, em paralelo, os resultados iriam demorar ainda mais para serem finalizados. Tendo isto em vista, parte dos experimentos foi executada em um processador Intel(R) Core(TM) i7 – 4500 CPU @ 1.8GHz, com 32 GB de RAM e executando sistema operacional Linux Mint 64-*bits*. Outra parte foi executada em um processador Intel(R) Core(TM) i7 – 4500 CPU @ 1.8GHz, com 16 GB de RAM e executando sistema operacional Linux Ubuntu Server 64-*bits*. Uma outra parte foi conduzida em um processador Intel(R) Core(TM) i7 – 7700 CPU @ 3.6GHz, com 32 GB de RAM e executando sistema operacional Linux Ubuntu 64-*bits*. A ultima parte foi obtida em um processador Intel(R) Core(TM) i7 – 8700 CPU @ 3.2GHz, com 16 GB de RAM e executando sistema operacional Linux Ubuntu 64-*bits*.

5.3 Descrição dos *Benchmarks*

Os *benchmarks* com os sistemas de controle usados para avaliar a abordagem proposta são descritos a seguir. Em particular, estes estão disponíveis na Tabela B, no Anexo B, e fornece informações sobre as matrizes de espaço de estados **A**, **B** e **C**, tempo de assentamento requerido t_{sr} e porcentagem de sobressinal requerido PO_{r}. Todos os *benchmarks* apresentam a matriz de espaço de estados $D = [0]$.

Esses sistemas foram escolhidos baseados na possibilidade de se ter diferentes cenários em relação aos autovalores da matriz **A**. Todos os experimentos foram conduzidos utilizando os mesmos valores das variáveis `ts=0.5`, `p=5`, `bounds_l=-0.5`, `bounds_u=0.5`, `popsize=100` e `nbgen=100`. Esses valores foram utilizados e definidos empiricamente através da realização de testes de caráter experimental e observou-se o desempenho da ferramenta usando esses valores e verificando que seriam bons valores base para realizar os experimentos do *benchmark*, para se conseguir sintetizar com sucesso. Existem sistemas de 2ª (*benchmarks* 1, 5, 8, 12 e 15), 3ª (*benchmarks* 2, 9 e 13), 4ª (*benchmarks* 3, 6, 7, 10 e 14) e 5ª (*benchmarks* 4 e 11) ordens. Foram escolhidos *benchmarks* com diferentes tipos de configurações de autovalores. Têm *ben-*

chmarks com a matriz **A** que possui autovalores todos iguais e de números reais. Outros com autovalores todos diferentes (reais e/ou complexos). Também existem sistemas com autovalores complexos. Com essa diversidade, pode-se abranger sistemas que apresentam comportamentos diferentes para se poder fazer uma análise mais completa. Além disso, o *benchmark* 15 representa uma planta que representa um motor DC. Os *benchmarks* utilizados nesse trabalho não contêm sistemas MIMO (*multiple-input multiple-output*), uma vez que a metodologia proposta é capaz de verificar apenas os SISO, no momento.

Como pode ser observado na Tabela B, há 45 experimentos considerando somente tempo de assentamento, divididos em 3 diferentes configurações de FWL (8, 16 e 32 *bits*), mais 45 experimentos considerando somente sobressinal, novamente divididos em 3 diferentes configurações de FWL (8, 16 e 32 *bits*). Também foram realizados 45 experimentos considerando ambas as propriedades simultaneamente (tempo de assentamento e sobressinal) e incluindo novamente experimentos de 3 diferentes formatos de FWL.

5.4 Resultados Experimentais

A Tabela 5.1 resume os resultados experimentais dos *benchmarks* descritos na Tabela B. A coluna **ID** descreve a identificação do *benchmark*, a segunda coluna representa a ordem do sistema de cada *benchmark*, **R** representa resultado da síntese (bem ou mal sucedida), **T** é o número de tentativas para realizar a síntese, e, finalmente, $\langle 4,4 \rangle$, $\langle 8,8 \rangle$ e $\langle 16,16 \rangle$ representam o formato de FWL considerado, onde representam no total 8, 16 e 32 *bits*, respectivamente. Pode-se perceber que quando há falhas na síntese, a Tabela 5.1 mostra os *status* `MA` e `TO` que significam que foi atingido o número máximo de tentativas pre-estabelecidas e houve um estouro do tempo de execução da síntese, respectivamente.

O processo de síntese executado pela ferramenta desenvolvida que aplica a metodologia explanada neste trabalho apresentou, no total, 101 casos de síntese bem-sucedida. Desse total, 39 foram com os experimentos considerando somente o tempo de assentamento, onde tiveram 13 casos de sucesso para cada uma das configurações de FWL (8, 16 e 32 *bits*). Adicionando a isso, 32 casos de sucesso foram com os experimento considerando somente sobressinal, onde 10 foi para a configuração com 8 *bits*, 11 com 16 *bits* e outros 11 com 32 *bits*. Completando, tiveram 10 casos de sucesso para cada configuração de FWL (8, 16 e 32 *bits*), totalizando mais 30 experimentos bem-sucedidos. Por outro lado, houve 34 casos em que a síntese foi mal-

Tabela 5.1: Resultados experimentais

ID	Ordem	Tempo de Assentamento						Sobressinal						Tempo de Assentamento & Sobressinal					
		<4,4>		<8,8>		<16,16>		<4,4>		<8,8>		<16,16>		<4,4>		<8,8>		<16,16>	
		R	***T***	***R***	***T***	***R***	***T***	***R***	***T***	***R***	***T***	***R***	***T***	***R***	***T***	***R***	***T***	***R***	***T***
1	2	S	1	S	1	S	3	S	5	S	1	S	1	S	1	S	1	S	3
2	3	S	1	S	1	S	4	F	TO	S	1	S	1	S	1	S	1	S	4
3	4	S	1	S	2	S	2	S	1	S	1	S	1	S	1	S	2	S	2
4	5	S	1	S	1	S	1	S	1	S	1	S	1	S	1	S	1	S	1
5	2	S	1	S	1	S	7	S	19	S	1	S	1	S	1	S	1	S	7
6	4	S	1	S	5	S	41	S	5	S	1	S	1	F	TO	F	TO	F	TO
7	4	S	1	S	1	S	1	F	TO	F	TO	F	TO	F	TO	F	TO	F	TO
8	2	S	1	S	1	S	3	S	3	S	1	S	1	S	1	S	1	S	1
9	3	S	1	S	1	S	3	S	1	S	1	S	1	S	22	S	1	S	3
10	4	S	3	S	2	S	6	S	1	S	1	S	1	S	3	S	2	S	2
11	5	S	1	S	1	S	1	S	1	S	1	S	1	S	12	S	1	S	1
12	2	F	MA	F	MA	F	MA	F	MA	F	MA	F	MA	F	MA	F	MA	F	MA
13	3	F	MA	F	MA	F	MA	F	MA	F	MA	F	MA	F	MA	F	MA	F	MA
14	4	S	1	S	1	S	1	F	TO	F	TO	F	TO	F	TO	F	TO	F	TO
15	2	S	1	S	1	S	1	S	1	S	1	S	1	S	1	S	1	S	1
S=Síntese bem-sucedida e F=Síntese mal-sucedida. MA=Número máximo de tentativas atingido. TO=Tempo esgotado																			

sucedida. Dentre esses casos, 18 tiveram esse resultado devido a atingirem o número máximo de tentativas pré-estabelecidas (`MAXTENT=200`) e 16 devido ao estouro do tempo de execução da síntese (por padrão, adotou-se 15 dias devido ao tempo que se tem disponível para executar os experimentos).

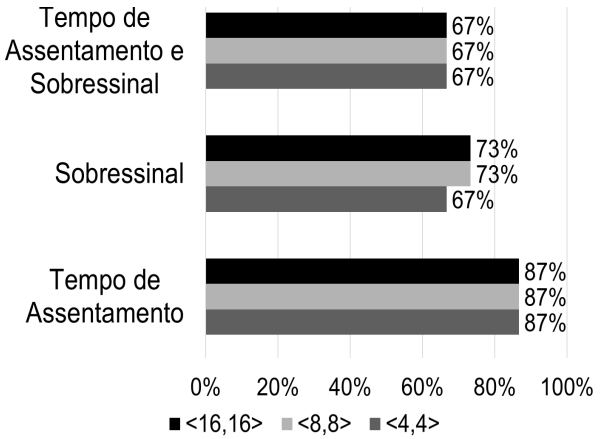


Figura 5.1: Gráfico do percentual de sucesso nos experimentos.

Como pode ser visto no gráfico da Figura 5.1, os experimentos de tempo de assentamento tiveram um percentual de sucesso de 87% para as 3 diferentes formatos de FWL ($\langle 4,4\rangle$, $\langle 8,8\rangle$ e $\langle 16,16\rangle$). Por outro lado, os experimentos de sobressinal tiveram um índice de síntese com sucesso de 73% para os formatos $\langle 8,8\rangle$ e $\langle 16,16\rangle$, e 67% para o formato $\langle 4,4\rangle$. Para os experimentos considerando tempo de assentamento e sobressinal sumultaneamente, obteve-se 67% para todos os formatos FWL analisados. Com isso, pode-se perceber que o índice de sucesso da síntese de controladores mostrou-se satisfatória, visto que uma grande quantidade de experimentos tiveram sucesso no processo de síntese.

Nos experimentos levando em conta somente tempo de assentamento, nota-se que quando aplicado os efeitos de FWL no formato $\langle 4,4\rangle$, a síntese foi realizada com sucesso com uma ou três tentativas (*benchmarks* 1, 2,3, 4, 5, 6, 7, 8, 9, 10, 11, 14 e 15), sendo que em todos esses casos precisaram somente de uma tentativa, e somente o *benchmank* 10 levou três tentativas. Com o formato de FWL $\langle 8,8\rangle$, os casos de síntese com sucesso, levaram um, dois ou cinco tentativas (*benchmarks* 1, 2,3, 4, 5, 6, 7, 8, 9, 10, 11, 14 e 15), sendo que somente dois *benchmarks* (3 e 10) precisaram de duas tentativas e um *benchmark* (6) realizou em sua quinta tentativa. Os experimentos com o formato de FWL $\langle 16,16\rangle$, tiveram um número maior, em média, que os outros formatos, onde levaram uma, duas, três, quatro, seis, sete e quarenta e uma tentativas para sintetizarem com sucesso (*benchmarks* 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 14 e 15).

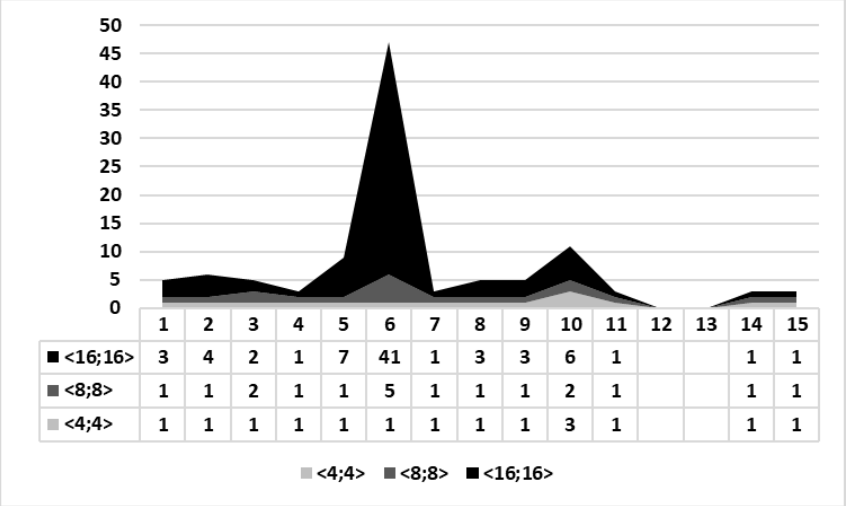
Considerando apenas sobressinal, os experimentos com o formato $\langle 4,4\rangle$ levaram um, três, cinco e dezenove tentativas para executar a síntese com sucesso (*benchmarks* 1,3, 4, 5, 6, 8, 9, 10, 11 e 15). Por outro lado, os demais formatos ($\langle 8,8\rangle$ e $\langle 16,16\rangle$) levaram apenas uma tentativa para se sintetizar com sucesso (*benchmarks* 1, 2, 3, 4, 5, 6, 8, 9, 10, 11 e 15), possivelmente devido ao fato de serem *benchmarks* com sistemas que possuem autovalores mais simples (grande maioria com autovalores puramente de números reais), sem muitas ondulações provocadas por autovalores complexos.

Quando realiza-se os experimentos considerando ambas as restrições (tempo de assentamento e sobressinal), no formato de FWL $\langle 4,4\rangle$ necessitaram-se de uma, três, doze e vinte e duas tentativas (*benchmarks* 1, 2, 3, 4, 5, 8, 9, 10, 11 e 15). Já no formato $\langle 8,8\rangle$, uma ou duas tentativas de se sintetizar com sucesso (*benchmarks* 1, 2, 3, 4, 5, 8, 9, 10, 11 e 15). E no formato $\langle 16,16\rangle$, os experimentos levaram uma, duas, três, quatro ou sete tentativas (*benchmarks* 1, 2, 3, 4, 5, 8, 9, 10, 11 e 15).

Nota-se que a ferramenta de síntese falhou na tentativa de sintetizar um controlador que atendesse aos requisitos dos *benchmarks* 12 e 13, seja considerando somente tempo de assentamento, sobressinal ou ambos simultaneamente, como pode ser visto na Tabela 5.1. Esses experimentos falharam devido a atingirem o limite máximo de tentativas estabelecido (`MAXTENT=200`). Uma das possíveis causas pode ser devido a possuírem sistemas instáveis em malha aberta e a metodologia ter dificuldade para encontrar um controlador que atenda a requisitos de estabilidade e desempenho (tempo de assentamento e/ou sobressinal), visto que o algoritmo genético pode não conseguir gerar candidatos que atendam a esses requisitos, a nível de problema de otimização.

Outra característica que pode ser percebida nos resultados da Tabela 5.1, é que a ferramenta também falhou na tentativa de síntese para os *benchmarks* 6, 7 e 14, devido a ultrapassarem o limite de tempo de execução, onde alguns deles passaram entre 15 e 30 dias em execução. Fato esse que pode ser justificado pela ferramenta de síntese desenvolvida não responder bem quando lida com sistemas que possuem seus autovalores complexos em malha fechada, como pode ser observado na Tabela B.1, no Anexo B.

Abaixo, é mostrado uma análise dos resultados obtidos durante os experimentos. A análise é feita através de gráficos de área, para tentar verificar o número de tentativas em cada experimento e pelo formato FWL utilizado. No eixo das ordenadas é mostrado o número de tentativas executados para sintetizar um controlador que atenda aos requisitos. No eixo das abcissas é descrito o **ID** dos experimentos, conforme a Tabela 5.1.

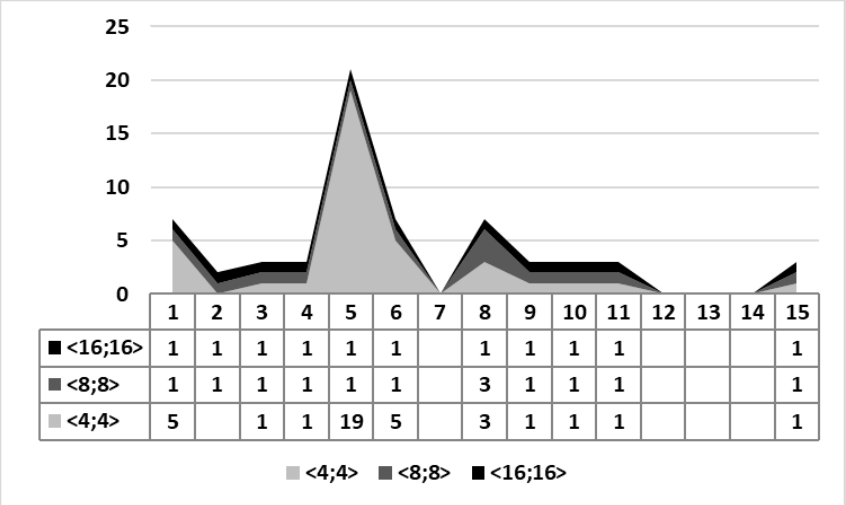


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
<16;16>	3	4	2	1	7	41	1	3	3	6	1			1	1
<8;8>	1	1	2	1	1	5	1	1	1	2	1			1	1
<4;4>	1	1	1	1	1	1	1	1	1	3	1			1	1

Figura 5.2: Gráfico de área dos experimentos de tempo de assentamento com a relação ao número de tentativas para síntese.

A Figura 5.2 mostra o desempenho dos experimentos considerando somente tempo de assentamento como requisito. É interessante notar uma tendência dos experimentos com o formato de FWL $\langle 16,16\rangle$ em necessitar de um maior número de tentativas para sintetizar um controlador, visto que para os demais formatos ($\langle 4,4\rangle$ e $\langle 8,8\rangle$), os experimentos demandam um menor número de tentativas. Neste caso, pode-se notar uma disposição de precisar de uma quantidade maior de tentativas, conforme se aumenta o número de *bits* do formato de FWL. Vale ressaltar que as lacunas na Figura 5.2 remete-se à ferramenta não ter conseguido sintetizar o experimento com sucesso, devido a atingirem o limite máximo de tentativas estabelecido.

Neste conjunto de experimentos, nota-se que o *benchmark* 6 precisou de 41 tentativas (com a configuração de FWL $\langle 16, 16 \rangle$), o que é uma quantidade alta, para conseguir sintetizar um controlador que atendesse ao requisito de tempo de assentamento, isso pode ser devido ao fato de o sistema em questão possuir autovalores, em sua matriz $\mathbf{A}$, no domínio dos números complexos, o que pode trazer essa dificuldade maior em sintetizá-lo. Observa-se que para uma quantidade de *bits* intermediária ($\langle 8, 8 \rangle$), demandou um número de tentativas intermediárias também, portanto, é notório que há uma influência da quantidade de *bits* na representação de FWL no número de tentativas para sintetizar controladores de sistemas de controle com respeito a tempo de assentamento.

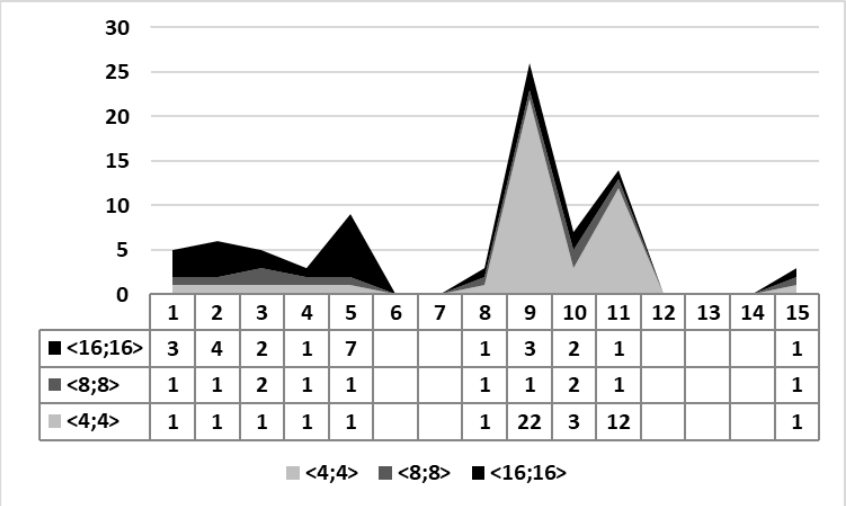


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
<16;16>	1	1	1	1	1	1		1	1	1	1				1
<8;8>	1	1	1	1	1	1		3	1	1	1				1
<4;4>	5		1	1	19	5		3	1	1	1				1

Figura 5.3: Gráfico de área dos experimentos de sobressinal com a relação ao número de tentativas para síntese.

Por outro lado, na Figura 5.3, os experimentos realizados considerando somente sobressinal, demandam, em média, um maior número de tentativas, para se realizar a síntese com sucesso, no formato de FWL $\langle 4, 4 \rangle$ e um menor número de tentativas com o formato $\langle 16, 16 \rangle$. A Figura 5.3 também mostra que os experimentos com o formato $\langle 8, 8 \rangle$, apresentam resultados que precisaram de uma quantidade intermediária de tentativas (entre os formatos $\langle 4, 4 \rangle$ e $\langle 16, 16 \rangle$). Portanto, nota-se essa tendência de quanto menor a quantidade de *bits* de FWL, maior foi o número de tentativas para se conseguir sintetizar os controladores. Novamente, os *benchmarks* 12 e 13 falharam devido a ultrapassarem o limite máximo de tentativas pré-estabelecido, o que pode ser justificado por esses sistemas possuírem polos que levam à instabilidade do sistema, em malha aberta. Pode-se também notar que o *benchmark* 5, para os experimentos de sobressi-

nal, demandou um grande número de tentativas para se conseguir sintetizar um controlador de forma a atender este requisito (19 tentativas com formato $\langle 4,4 \rangle$), fato este caracterizado com sistemas que possuem autovalores no domínio dos números complexos, em malha aberta. Diferentemente dos experimentos considerando somente tempo de assentamento, o *benchmark* 5 teve um número maior de tentativas para quantidade menor de *bits*.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
<16;16>	3	4	2	1	7			1	3	2	1				1
<8;8>	1	1	2	1	1			1	1	2	1				1
<4;4>	1	1	1	1	1			1	22	3	12				1

Figura 5.4: Gráfico de área dos experimentos de tempo de assentamento e sobressinal juntos, com a relação ao número de tentativas para síntese.

Nos experimentos onde se consideram ambos os requisitos (tempo de assentamento e sobressinal), a análise dos dados é diferente dos dois anteriores, individualmente. A Figura 5.4 mostra que, em média, os experimentos entre 1 e 5 tem um comportamento parecido com os da Figura 5.2. Por outro lado, os experimentos entre 8 e 12 apresentam comportamento semelhante aos da Figura 5.3. Uma importante característica que pode ser notada no gráfico da Figura 5.4, dentro de um contexto geral e em média, é que as características dos experimentos de sobressinal (menos *bits* e mais tentativas) foram dominantes quando comparadas com os experimentos de tempo de assentamento.

5.4.1 Considerações Finais

A metodologia desenvolvida neste trabalho que visa sintetizar controladores digitais com respeito a requisitos de máximo tempo de assentamento e sobressinal, aplicada aos *benchmarks* descritos na Tabela B, sintetizou, aproximadamente, 75% dos *benchmarks* com sucesso,

e falhou em 25% deles por atingir o número máximo de tentativas utilizado (`MAXTENT=200`) ou por tempo de execução esgotado, ou seja, demandou tempo excessivo.

A metodologia é capaz de realizar a síntese na grande maioria dos casos, ilustrados na Tabela B, principalmente os sistemas que já são estáveis em malha aberta, pois os que não são, a ferramenta teve dificuldade em atender a todos os requisitos, falhando no processo de síntese, devido a atingirem o limite máximo de tentativas. A ferramenta também não apresentou um desempenho muito bom quando tratou de sistemas com seus polos localizados no domínio dos números complexos, possivelmente devido a possuírem mais oscilações (taxa de amortecimento) e durante a quantificação pode causar um comportamento que pode levar a falha.

Portanto, os resultados apresentados mostram que a metodologia proposta é apropriada para sintetizar controladores digitais que atendam a requisitos de máximo tempo de assentamento e sobressinal, considerando aspectos de implementação (efeitos de FWL). Logo, a metodologia proposta pode ser útil para auxiliar engenheiros durante o estágio de projeto de controladores digitais para sistemas de controle implantados em uma plataforma física (microprocessada).

5.5 Resumo

Este capítulo apresentou a avaliação experimental realizada para a metodologia proposta neste trabalho, assim como considerações a serem feitas sobre a metodologia apresentada. Os experimentos foram conduzidos em computadores com diferentes configurações e os dados obtidos foram analisados, mostrando a viabilidade da metodologia para sintetizar controladores digitais, tendo um total de aproximadamente 75% de controladores sintetizados com sucesso, considerando todos os *benchmarks* propostos. A metodologia proposta mostrou-se útil para projetar controladores digitais, visto que além de atender requisitos de desempenho, também considera os efeitos de FWL sobre os controladores.

Capítulo 6

Conclusões

6.1 Considerações Finais

Esta dissertação assume como objetivos o desenvolvimento de uma metodologia de síntese de controladores digitais que atenda a requisitos de desempenho de sistemas de controle com respeito a resposta ao degrau (tempo de assentamento e sobressinal) e requisitos de implementação, mais precisamente os efeitos de FWL, e também, como parte da síntese mas que pode ser usado de forma isolada, o desenvolvimento de uma metodologia de verificação de controladores digitais considerando esses aspectos de desempenho e implementação.

Para tal, o desenvolvimento das metodologias de síntese e verificação apoiou-se em alguns algoritmos mostrados neste trabalho que ajudam no processo de verificação formal de requisitos de desempenho, considerando os efeitos de FWL nos controladores digitais. Para a síntese, tais algoritmos são utilizados no processo de verificação que é uma das etapas da síntese a qual foi baseada na síntese denominada CEGIS e como outro módulo desse esquema de síntese, foi utilizado algoritmo genético para geração de candidatos.

Realizou-se em primeiro lugar uma revisão da literatura. Sobre a verificação de sistemas de controle, apresentaram-se algumas abordagens de verificação, porém deixaram de considerar aspectos cruciais que a presente dissertação buscou resolver, tais como a verificação de requisitos de desempenho em sistemas de controle discretos, considerando os efeitos de FWL nos controladores. Já sobre a síntese de sistemas de controle, foram encontrados poucos trabalhos relacionados, entretanto, dentre eles, alguns não consideravam também sistemas de controle discreto, bem como os requisitos de desempenho e os efeitos de FWL.

A metodologia de verificação de sistemas de controle com respeito a tempo de assentamento utilizou uma invariante, criada para o escopo desta dissertação, para realizar a verificação desta propriedade. Este método de verificação foi capaz de verificar sistemas de controle de forma eficaz, como mostram os resultados ilustrados no Capítulo 5, onde a verificação é parte do processo de síntese. O mesmo ocorre com a verificação com respeito a sobressinal, porém utilizou-se um algoritmo que estima o máximo pico para calcular o sobressinal com o objetivo de auxiliar na verificação dessa propriedade.

Como mostrado no Capítulo 5, a metodologia de síntese implementada na ferramenta DSVerifier foi capaz de sintetizar controladores digitais em uma diversidade de *benchmarks* de sistemas de controle especialmente criados para o presente trabalho, sendo este também uma contribuição. Este conjunto de *benchmarks* pode ser utilizado como um *data set* de teste para o uso em diferentes aplicações. Utilizou-se a técnica de CEGIS como método de síntese, de forma iterativa onde a cada iteração um controlador candidato foi gerado com o auxílio de algoritmo genético, como elucidado anteriormente.

As avaliações experimentais realizadas neste trabalho visam mostrar a aplicação da metodologia de síntese de controladores digitais que atendam a requisitos de máximo tempo de assentamento, sobressinal ou ambos simultaneamente, considerando os efeitos de FWL. Verificou-se que a metodologia desenvolvida neste trabalho, implementada em C++, como uma nova funcionalidade do DSVerifier funcionou bem para a maioria dos *benchmarks* utilizados. Porém, notou-se que a síntese não foi realizada com sucesso para alguns casos, devido a atingirem o limite máximo de tentativas ou por ter estourado o tempo de execução pré-estabelecido. Espera-se que estes resultados possam ser utilizados como referência em outras pesquisas para fins de comparação com outros métodos.

6.2 Propostas para Trabalhos Futuros

Em trabalhos futuros a pesquisa sugere a extensão da abordagem. As propostas podem ser divididas em dois tipos de melhorias. A primeira é relacionada a parte de verificação e a segunda a síntese em si. Considera-se importante serem propostas melhorias nos algoritmos de verificação utilizando técnicas de aprendizagem de máquinas, por exemplo. Outra melhoria está relacionada com formulações mais robustas de geração de candidatos no esquema de síntese. Essas abordagens podem utilizar técnicas como aprendizagem de máquinas ou redes neurais

que são ferramentas que podem ajudar na melhoria do processo de geração de candidatos, visto que neste trabalho, existem alguns casos em que a metodologia não funcionou bem. Também se propõe uma adaptação na metodologia que visa suportar sistemas MIMO. Outra coisa que pode fortalecer a pesquisa é a realização de uma análise comparativa com algumas técnicas tradicionais de controle relacionadas a projeto de controladores.

Bibliografia

[1] BOYD, S.; BARRATT, C. *Linear controller design: limits of performance*. 1991.

[2] SIMONS, R. *Levers of control: how managers use innovative control systems to drive strategic renewal*. : Harvard Business Press, 1994.

[3] HASPESLAGH, D. R.; MOERMAN, E. *Digital sigma-delta modulator and arithmetic overflow protected adder*. : Google Patents, 1994. US Patent 5,325,399.

[4] ISTEPANIAN, R.; WHIDBORNE, J. F. *Digital controller implementation and fragility: A modern perspective*. : Springer Science & Business Media, 2012.

[5] BESSA, I. V. de; ISMAIL, H.; CORDEIRO, L. C.; FILHO, J. E. C. Verification of fixed-point digital controllers using direct and delta forms realizations. *Design Autom. for Emb. Sys.*, v. 20, n. 2, p. 95–126, 2016.

[6] BESSA, I.; ISMAIL, H.; PALHARES, R.; CORDEIRO, L.; FILHO, J. E. C. Formal non-fragile stability verification of digital control systems with uncertainty. *IEEE Transactions on Computers*, IEEE, v. 66, n. 3, p. 545–552, 2017.

[7] KEEL, L. H.; BHATTACHARYYA, S. P. Robust, fragile, or optimal? *IEEE Trans. Autom. Control*, v. 42, n. 8, p. 1098–1105, Aug 1997. ISSN 0018-9286.

[8] YORDANOV, B.; TUMOVA, J.; CERNA, I.; BARNAT, J.; BELTA, C. Temporal logic control of discrete-time piecewise affine systems. *IEEE Transactions on Automatic Control*, IEEE, v. 57, n. 6, p. 1491–1504, 2012.

[9] CHAVES, L. C.; ISMAIL, H. I.; BESSA, I. V.; CORDEIRO, L. C.; FILHO, E. B. de L. Verifying fragility in digital systems with uncertainties using dsverifier v2.0. *Journal of Systems and Software*, v. 153, p. 22–43, 2019.

[10] BESSA, I.; ABREU, R. B.; FILHO, J. E. C.; CORDEIRO, L. C. Smt-based bounded model checking of fixed-point digital controllers. In: *IECON 2014 - 40th Annual Conference of the IEEE Industrial Electronics Society, Dallas, TX, USA, October 29 - November 1, 2014*. 2014. p. 295–301.

[11] ABREU, R. B.; GADELHA, M. Y. R.; CORDEIRO, L. C.; FILHO, E. B. de L.; JR., W. S. da S. Bounded model checking for fixed-point digital filters. *J. Braz. Comput. Soc.*, Springer London, v. 22, n. 1, p. 1, 2016.

[12] BELTA, C. Formal methods for dynamical systems. In: IEEE. *TIME*. 2014. p. 3–3.

[13] RACCHETTI, L.; FANTUZZI, C.; TACCONI, L. Verification and validation based on the generation of testing sequences from timing diagram specifications in industrial automation. In: IEEE. *IECON*. 2015. p. 002816–002821.

[14] NILSSON, P.; HUSSIEN, O.; BALKAN, A.; CHEN, Y.; AMES, A. D.; GRIZZLE, J. W.; OZAY, N.; PENG, H.; TABUADA, P. Correct-by-construction adaptive cruise control: Two approaches. *IEEE Trans. Contr. Sys. Techn.*, IEEE, v. 24, n. 4, p. 1294–1307, 2016.

[15] CHAVES, L.; BESSA, I.; ISMAIL, H.; FRUTUOSO, A. B. dos S.; CORDEIRO, L. C.; FILHO, E. B. de L. Dsverifier-aided verification applied to attitude control software in unmanned aerial vehicles. *IEEE Trans. Reliability*, v. 67, n. 4, p. 1420–1441, 2018.

[16] FRANKLIN, G. F.; POWELL, J. D.; WORKMAN, M. L. *Digital control of dynamic systems*. : Addison-wesley Menlo Park, CA, 1998.

[17] GROSS, K. H. Formal specification and analysis approaches for spacecraft attitude control requirements. In: IEEE. *Aerospace Conference*. 2017. p. 1–11.

[18] JIN, X.; DONZÉ, A.; DESHMUKH, J. V.; SESHIA, S. A. Mining requirements from closed-loop control models. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, IEEE, v. 34, n. 11, p. 1704–1717, 2015.

[19] LI, G. On pole and zero sensitivity of linear systems. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, IEEE, v. 44, n. 7, p. 583–590, 1997.

[20] WALSH, M.; DEVEREAUX, P. J.; GARG, A. X.; KURZ, A.; TURAN, A.; RODSETH, R. N.; CYWINSKI, J.; THABANE, L.; SESSLER, D. I. Relationship between intraoperative

mean arterial pressure and clinical outcomes after noncardiac surgerytoward an empirical definition of hypotension. *Anesthesiology: The Journal of the American Society of Anesthesiologists*, The American Society of Anesthesiologists, v. 119, n. 3, p. 507–515, 2013.

[21] JHA, S.; GULWANI, S.; SESHIA, S. A.; TIWARI, A. Oracle-guided component-based program synthesis. In: ACM. *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*. 2010. p. 215–224.

[22] RAVANBAKHSH, H.; SANKARANARAYANAN, S. Counter-example guided synthesis of control lyapunov functions for switched systems. In: IEEE. *Decision and Control (CDC), 2015 IEEE 54th Annual Conference on*. 2015. p. 4232–4239.

[23] ABATE, A.; DAVID, C.; KESSELI, P.; KROENING, D.; POLGREEN, E. Counterexample guided inductive synthesis modulo theories. In: SPRINGER. *International Conference on Computer Aided Verification*. 2018. p. 270–288.

[24] OGATA, K.; YANG, Y. *Modern control engineering*. : Prentice Hall, 2002.

[25] DEB, K. Introduction to genetic algorithms for engineering optimization. In: *New Optimization Techniques in Engineering*. : Springer, 2004. p. 13–51.

[26] ISMAIL, H. I.; BESSA, I. V.; CORDEIRO, L. C.; FILHO, E. B. de L.; FILHO, J. E. C. Dsverifier: A bounded model checking tool for digital systems. In: *Model Checking Software*. : Springer, 2015. p. 126–131.

[27] CHAVES, L.; BESSA, I.; CORDEIRO, L. C.; KROENING, D.; FILHO, E. B. de L. Verifying digital systems with MATLAB. In: *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis, Santa Barbara, CA, USA, July 10 - 14, 2017*. : ACM, 2017. p. 388–391.

[28] JHALA, R.; MAJUMDAR, R. Software model checking. *ACM Computing Surveys (CSUR)*, ACM, v. 41, n. 4, p. 21, 2009.

[29] DOROFEEVA, R.; EL-FAKIH, K.; MAAG, S.; CAVALLI, A. R.; YEVTUSHENKO, N. Fsm-based conformance testing methods: A survey annotated with experimental evaluation. *Information and Software Technology*, Elsevier, v. 52, n. 12, p. 1286–1297, 2010.

[30] CHENG, K. C.; YAP, R. H. An mdd-based generalized arc consistency algorithm for positive and negative table constraints and some global constraints. *Constraints*, Springer, v. 15, n. 2, p. 265–304, 2010.

[31] BALARIN, F.; SANGIOVANNI-VINCENTELLI, A. L. An iterative approach to language containment. In: SPRINGER. *International Conference on Computer Aided Verification*. 1993. p. 29–40.

[32] ALHAZOV, A.; MARTÍN-VIDE, C.; PAN, L. Solving a pspace-complete problem by recognizing p systems with restricted active membranes. *Fundamenta Informaticae*, IOS Press, v. 58, n. 2, p. 67–77, 2003.

[33] BENEŠ, N.; BEZDĚK, P.; LARSEN, K. G.; SRBA, J. Language emptiness of continuous-time parametric timed automata. In: SPRINGER. *International Colloquium on Automata, Languages, and Programming*. 2015. p. 69–81.

[34] TAŞIRAN, S.; HOJATI, R.; BRAYTON, R. K. Language containment of non-deterministic ω-automata. In: SPRINGER. *Advanced Research Working Conference on Correct Hardware Design and Verification Methods*. 1995. p. 261–277.

[35] JR, E. M. C.; GRUMBERG, O.; KROENING, D.; PELED, D.; VEITH, H. *Model checking*. : MIT press, 2018.

[36] BIERE, A.; CIMATTI, A.; CLARKE, E.; STRICHMAN, O.; ZHU, Y. Bounded Model Checking. *Advances in Computers*, p. 117–148, 2003.

[37] ROCHA, H.; ISMAIL, H.; CORDEIRO, L. C.; BARRETO, R. S. Model checking embedded C software using k-induction and invariants. In: *2015 Brazilian Symposium on Computing Systems Engineering, SBESC 2015, Foz do Iguacu, Brazil, November 3-6, 2015*. : IEEE Computer Society, 2015. p. 90–95.

[38] ROCHA, W.; ROCHA, H.; ISMAIL, H.; CORDEIRO, L. C.; FISCHER, B. Depthk: A k-induction verifier based on invariant inference for C programs - (competition contribution). In: *Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part II*. 2017. (Lecture Notes in Computer Science, v. 10206), p. 360–364.

[39] GADELHA, M. Y. R.; MONTEIRO, F. R.; CORDEIRO, L. C.; NICOLE, D. A. ESBMC v6.0: Verifying C programs using k-induction and invariant inference - (competition contribution). In: *Tools and Algorithms for the Construction and Analysis of Systems - 25 Years of TACAS: TOOLympics, Held as Part of ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part III.* : Springer, 2019. (Lecture Notes in Computer Science, v. 11429), p. 209–213.

[40] KRKA, I.; BRUN, Y.; POPESCU, D.; GARCIA, J.; MEDVIDOVIC, N. Using dynamic execution traces and program invariants to enhance behavioral model inference. In: ACM. *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2*. 2010. p. 179–182.

[41] GADELHA, M. Y. R.; ISMAIL, H. I.; CORDEIRO, L. C. Handling Loops in Bounded Model Checking of C Programs via k-Induction. *Software Tools for Technology Transfer*, p. 97–114, 2017.

[42] GADELHA, M. R.; MONTEIRO, F.; CORDEIRO, L.; NICOLE, D. Esbmc v6. 0: Verifying c programs using k-induction and invariant inference. In: SPRINGER. *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. 2019. p. 209–213.

[43] ROCHA, W.; ROCHA, H.; ISMAIL, H.; CORDEIRO, L.; FISCHER, B. Depthk: A k-induction verifier based on invariant inference for c programs. In: SPRINGER. *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. 2017. p. 360–364.

[44] MORSE, J.; RAMALHO, M.; CORDEIRO, L.; NICOLE, D.; FISCHER, B. ESBMC 1.22. In: *TACAS, LNCS*. 2014. v. 8413, p. 405–407.

[45] BIERE, A.; CIMATTI, A.; CLARKE, E. M.; STRICHMAN, O.; ZHU, Y. et al. Bounded model checking. *ADV COMPUT*, v. 58, n. 11, p. 117–148, 2003.

[46] SHEERAN, M.; SINGH, S.; STÅLMARCK, G. Checking safety properties using induction and a sat-solver. In: SPRINGER. *FMCAD*. 2000. p. 127–144.

[47] GADELHA, M. Y. R.; ISMAIL, H. I.; CORDEIRO, L. C. Handling loops in bounded model checking of C programs via k-induction. *STTT*, v. 19, n. 1, p. 97–114, 2017.

[48] KATOK, A.; HASSELBLATT, B. *Introduction to the modern theory of dynamical systems*. : Cambridge university press, 1995.

[49] KAILATH, T. *Linear systems*. : Prentice-Hall Englewood Cliffs, NJ, 1980.

[50] BAKSHI, U.; BAKSHI, V. *Control System Engineering*. : Technical Publications Pune, 2008.

[51] ROSE, N. J. Linear algebra and its applications (gilbert strang). *SIAM Review*, SIAM, v. 24, n. 4, p. 499–501, 1982.

[52] WEYL, H. *Symmetry*. : Princeton University Press, 2015.

[53] COXETER, H. S. M. *Non-euclidean geometry*. : Cambridge University Press, 1998.

[54] WILLSKY, A. S.; YOUNG, I. T. *Signals and systems*. : Prentice-Hall International, 1997.

[55] OPPENHEIM, A. V. *Discrete-time signal processing*. : Pearson Education India, 1999.

[56] ASTRÖM, K. J.; MURRAY, R. M. *Feedback systems: an introduction for scientists and engineers*. : Princeton university press, 2010.

[57] HAYKIN, S.; VEEN, B. V. *Signals and systems*. : John Wiley & Sons, 2007.

[58] CHEN, C.-T. *Linear system theory and design*. : Oxford University Press, Inc., 1995.

[59] SMITH, S. W. et al. The scientist and engineer's guide to digital signal processing. California Technical Pub. San Diego, 1997.

[60] BITTANTI, S.; LAUB, A. J.; WILLEMS, J. C. *The Riccati Equation*. : Springer Science & Business Media, 2012.

[61] LOAN, C. V. A symplectic method for approximating all the eigenvalues of a hamiltonian matrix. *Linear Algebra and its Applications*, Elsevier, v. 61, p. 233–251, 1984.

[62] ALUR, R.; BODIK, R.; JUNIWAL, G.; MARTIN, M. M.; RAGHOTHAMAN, M.; SESHIA, S. A.; SINGH, R.; SOLAR-LEZAMA, A.; TORLAK, E.; UDUPA, A. Syntax-guided synthesis. In: IEEE. *Formal Methods in Computer-Aided Design (FMCAD), 2013*. 2013. p. 1–8.

[63] ZHOU, J.; ZHANG, C.; WEN, C. Robust adaptive output control of uncertain nonlinear plants with unknown backlash nonlinearity. *IEEE Transactions on automatic control*, IEEE, v. 52, n. 3, p. 503–509, 2007.

[64] ZHOU, K.; DOYLE, J. C.; GLOVER, K. et al. *Robust and optimal control*. : Prentice hall New Jersey, 1996.

[65] COHEN, N. H. *Ada as a second language*. : McGraw-Hill Higher Education, 1995.

[66] CHANG, K. H.; BLISS, W. G. Finite word-length effects of pipelined recursive digital filters. *IEEE Transactions on Signal Processing*, IEEE, v. 42, n. 8, p. 1983–1995, 1994.

[67] HAUSER, J. R. Handling floating-point exceptions in numeric programs. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, ACM, v. 18, n. 2, p. 139–174, 1996.

[68] WANG, T. E.; GAROCHE, P.; ROUX, P.; JOBREDEAUX, R.; FERON, E. Formal analysis of robustness at model and code level. In: *HSCC*. : ACM, 2016. p. 125–134.

[69] SADRADDINI, S.; BELTA, C. Formal methods for adaptive control of dynamical systems. *arXiv preprint arXiv:1703.07704*, 2017.

[70] HASSANI, K.; LEE, W.-S. Multi-objective design of state feedback controllers using reinforced quantum-behaved particle swarm optimization. *Applied Soft Computing*, Elsevier, v. 41, p. 66–76, 2016.

[71] RAVANBAKHSH, H.; SANKARANARAYANAN, S. Robust controller synthesis of switched systems using counterexample guided framework. In: IEEE. *2016 international conference on embedded software (EMSOFT)*. 2016. p. 1–10.

[72] ABATE, A.; BESSA, I.; CATTARUZZA, D.; CORDEIRO, L. C.; DAVID, C.; KESSELI, P.; KROENING, D. Sound and automated synthesis of digital stabilizing controllers for continuous plants. In: *Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control, HSCC 2017, Pittsburgh, PA, USA, April 18-20, 2017*. : ACM, 2017. p. 197–206.

[73] ABATE, A.; BESSA, I.; CATTARUZZA, D.; CORDEIRO, L. C.; DAVID, C.; KESSELI, P.; KROENING, D.; POLGREEN, E. Automated formal synthesis of digital controllers for

state-space physical plants. In: *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*. : Springer, 2017. (Lecture Notes in Computer Science, v. 10426), p. 462–482.

[74] ABATE, A.; BESSA, I.; CATTARUZZA, D.; CHAVES, L.; CORDEIRO, L. C.; DAVID, C.; KESSELI, P.; KROENING, D.; POLGREEN, E. Dssynth: an automated digital controller synthesis tool for physical plants. In: *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2017, Urbana, IL, USA, October 30 - November 03, 2017*. : IEEE Computer Society, 2017. p. 919–924.

[75] VERDIER, C. F.; MAZO, M. Formal synthesis of analytic controllers for sampled-data systems via genetic programming. In: IEEE. *2018 IEEE Conference on Decision and Control (CDC)*. 2018. p. 4896–4901.

[76] SPRINGER, T. A. *Invariant theory*. : Springer, 2006.

[77] DORF, R. C.; BISHOP, R. H. *Modern control systems*. : Pearson, 2011.

[78] GUENNEBAUD, G.; JACOB, B. et al. Eigen. *URl: http://eigen.tuxfamily.org*, 2010.

[79] MALLET, O. Galgo: Genetic algorithm in c++ with template metaprogramming and abstraction for constrained optimization. *URl: https://github.com/olmallet81/GALGO-2.0*, 2017.

[80] CAVALCANTE, T. R. F.; BESSA, I. V. de; CORDEIRO, L. C. Planning and evaluation of uav mission planner for intralogistics problems. In: IEEE. *2017 VII Brazilian Symposium on Computing Systems Engineering (SBESC)*. 2017. p. 9–16.

Apêndice A

Publicações

A.1 Referente à Pesquisa

- **Cavalcante, T. R. F.**, de Bessa, I. V., Cordeiro, L. C., Filho, E. L. *Formal Verification of Non-fragile Settling-Time and Overshoot Requirements for Digital State-Feedback Control Systems*. Em *Journal of Control, Automation and Electrical Systems* (JCAES), 2019. (**Submetido**)

A.2 Contribuições em outras Pesquisas

- **Cavalcante, T. R. F.**, Alves, E. H. S., Carvalho, C. B. *Radio Communication to Control and Run an Autonomous Mission for UAVs via a Mobile Application*. Em IV Escola Regional de Informática (ERIN), 2017. (**Publicado**)

- **Cavalcante, T. R. F.**, de Bessa, I. V., Cordeiro, L. C. *Planning and Evaluation of UAV Mission Planner for Intralogistics Problems*. Em VII Simpósio Brasileiro de Engenharia de Sistemas Computacionais (SBESC), 2017. (pp. 9-16). IEEE. (**Publicado**) [80]

Apêndice B

Benchmarks Utilizados

Tabela B.1: Descrição dos *benchmarks* usados nos experimentos do trabalho.

ID	A	B	C	t_{sr} **(s)**	PO_{r}
1	$\begin{bmatrix} -0.5 & 1.0 \\ 0.0 & -0.5 \end{bmatrix}$	$\begin{bmatrix} 0.0 \\ 2.5 \end{bmatrix}$	$\begin{bmatrix} 0.0 & 2.6 \end{bmatrix}$	2.5	5
2	$\begin{bmatrix} -0.5 & 1.0 & 0.0 \\ 0.0 & -0.5 & 1.0 \\ 0.0 & 0.0 & -0.5 \end{bmatrix}$	$\begin{bmatrix} -0.4 \\ 2.5 \\ -0.8 \end{bmatrix}$	$\begin{bmatrix} 0.0 & 2.6 & 0.0 \end{bmatrix}$	3.5	5
3	$\begin{bmatrix} -0.5 & 1.0 & 0.0 & 0.0 \\ 0.0 & -0.5 & 1.0 & 0.0 \\ 0.0 & 0.0 & -0.5 & 1.0 \\ 0.0 & 0.0 & 0.0 & -0.5 \end{bmatrix}$	$\begin{bmatrix} 1.0 \\ 2.5 \\ 1.0 \\ 1.0 \end{bmatrix}$	$\begin{bmatrix} 0.0 & 2.6 & 1.2 & 0.0 \end{bmatrix}$	4.5	30
4	$\begin{bmatrix} -0.5 & 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & -0.5 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & -0.5 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & -0.5 & 1.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & -0.5 \end{bmatrix}$	$\begin{bmatrix} -0.4 \\ -0.6 \\ 5.5 \\ 1.0 \\ -0.3 \end{bmatrix}$	$\begin{bmatrix} 0.0 & 2.6 & 0.5 & 1.2 & 0.0 \end{bmatrix}$	5.5	20
5	$\begin{bmatrix} -0.5 & 0.4 \\ -0.4 & -0.5 \end{bmatrix}$	$\begin{bmatrix} 0.0 \\ 2.5 \end{bmatrix}$	$\begin{bmatrix} 0.0 & 2.6 \end{bmatrix}$	3.0	5
6	$\begin{bmatrix} -0.5 & 0.4 & 1.0 & 0.0 \\ -0.4 & -0.5 & 0.0 & 1.0 \\ 0.0 & 0.0 & -0.5 & 0.4 \\ 0.0 & 0.0 & -0.4 & -0.5 \end{bmatrix}$	$\begin{bmatrix} 0.0 \\ 0.0 \\ 2.5 \\ 1.6 \end{bmatrix}$	$\begin{bmatrix} 0.0 & 2.6 & 0.0 & 2.0 \end{bmatrix}$	5.0	30
7	$\begin{bmatrix} -0.5 & 0.4 & 0.0 & 0.0 \\ -0.4 & -0.5 & 0.0 & 0.0 \\ 0.0 & 0.0 & -0.8 & 0.4 \\ 0.0 & 0.0 & -0.4 & -0.8 \end{bmatrix}$	$\begin{bmatrix} -0.4 \\ -0.6 \\ 2.5 \\ 1.6 \end{bmatrix}$	$\begin{bmatrix} 0.0 & 2.6 & 0.0 & 2.0 \end{bmatrix}$	10.0	4
8	$\begin{bmatrix} -0.2 & 0.0 \\ 0.0 & -0.3 \end{bmatrix}$	$\begin{bmatrix} -0.6 \\ 2.5 \end{bmatrix}$	$\begin{bmatrix} 0.0 & 2.6 \end{bmatrix}$	1.5	8
9	$\begin{bmatrix} -0.2 & 0.0 & 0.0 \\ 0.0 & -0.3 & 0.0 \\ 0.0 & 0.0 & -0.7 \end{bmatrix}$	$\begin{bmatrix} -0.8 \\ -0.7 \\ -0.5 \end{bmatrix}$	$\begin{bmatrix} 0.0 & 2.6 & 0.0 \end{bmatrix}$	2.0	8
10	$\begin{bmatrix} -0.2 & 0.0 & 0.0 & 0.0 \\ 0.0 & -0.3 & 0.0 & 0.0 \\ 0.0 & 0.0 & -0.7 & 0.0 \\ 0.0 & 0.0 & 0.0 & -0.9 \end{bmatrix}$	$\begin{bmatrix} -0.4 \\ 2.5 \\ 1.0 \\ -0.7 \end{bmatrix}$	$\begin{bmatrix} 0.0 & 2.6 & 1.2 & 0.0 \end{bmatrix}$	5.0	30
11	$\begin{bmatrix} -0.2 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & -0.3 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & -0.7 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & -0.9 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & -0.5 \end{bmatrix}$	$\begin{bmatrix} -0.5 \\ -0.2 \\ 2.5 \\ 2.0 \\ -0.8 \end{bmatrix}$	$\begin{bmatrix} 0.0 & 2.6 & 0.5 & 1.2 & 0.0 \end{bmatrix}$	10.0	18
12	$\begin{bmatrix} 4.5 & 1.0 \\ 0.0 & 4.5 \end{bmatrix}$	$\begin{bmatrix} 0.0 \\ 2.5 \end{bmatrix}$	$\begin{bmatrix} 0.0 & 2.6 \end{bmatrix}$	8.0	10
13	$\begin{bmatrix} 1.5 & 1.0 & 0.0 \\ 0.0 & 1.5 & 1.0 \\ 0.0 & 0.0 & 1.5 \end{bmatrix}$	$\begin{bmatrix} -0.4 \\ 2.5 \\ -0.8 \end{bmatrix}$	$\begin{bmatrix} 0.0 & 2.6 & 0.0 \end{bmatrix}$	10.0	9
14	$\begin{bmatrix} -0.5 & 0.4 & 0.0 & 0.0 \\ -0.4 & -0.5 & 0.0 & 0.0 \\ 0.0 & 0.0 & -0.8 & 0.4 \\ 0.0 & 0.0 & -0.4 & -0.8 \end{bmatrix}$	$\begin{bmatrix} -0.4 \\ -0.6 \\ -0.4 \\ -0.6 \end{bmatrix}$	$\begin{bmatrix} 0.0 & 2.6 & 0.0 & 2.0 \end{bmatrix}$	10.0	2
15	$\begin{bmatrix} -0.6386 & 0.4151 \\ -0.0098 & 0.8266 \end{bmatrix}$	$\begin{bmatrix} 0.2442 \\ 1.0745 \end{bmatrix}$	$\begin{bmatrix} 0.1807 & 0.2076 \end{bmatrix}$	6.5	5