sid.inpe.br/mtc-m21c/2020/04.04.03.50-TDI

UMA APRECIAÇÃO DO CONCENTRADOR DE DADOS REMOTO (RDC) EM UMA ARQUITETURA IMA DISTRIBUÍDA (DIMA) APLICADO AO CONTROLE DE ATITUDE DA PMM EM MODO NORMAL

Palmira dos Santos

Dissertação de Mestrado do Curso de Pós-Graduação em Engenharia e Tecnologia Espaciais/Mecânica Espacial e Controle, orientada pelo Dr. Marcelo Lopes de Oliveira e Souza, aprovada em 15 de abril de 2020.

URL do documento original:
<http://urlib.net/8JMKD3MGP3W34R/429ATB5>

INPE
São José dos Campos
2020

PUBLICADO POR:

Instituto Nacional de Pesquisas Espaciais - INPE
Gabinete do Diretor (GBDIR)
Serviço de Informação e Documentação (SESID)
CEP 12.227-010
São José dos Campos - SP - Brasil
Tel.:(012) 3208-6923/7348
E-mail: pubtc@inpe.br

sid.inpe.br/mtc-m21c/2020/04.04.03.50-TDI

UMA APRECIAÇÃO DO CONCENTRADOR DE DADOS REMOTO (RDC) EM UMA ARQUITETURA IMA DISTRIBUÍDA (DIMA) APLICADO AO CONTROLE DE ATITUDE DA PMM EM MODO NORMAL

Palmira dos Santos

Dissertação de Mestrado do Curso de Pós-Graduação em Engenharia e Tecnologia Espaciais/Mecânica Espacial e Controle, orientada pelo Dr. Marcelo Lopes de Oliveira e Souza, aprovada em 15 de abril de 2020.

URL do documento original:
<http://urlib.net/8JMKD3MGP3W34R/429ATB5>

INPE
São José dos Campos
2020

Dados Internacionais de Catalogação na Publicação (CIP)

Santos, Palmira dos.

Sa59a Uma apreciação do concentrador de dados remoto (RDC) em uma arquitetura IMA distribuída (DIMA) aplicado ao controle de atitude da PMM em modo normal / Palmira dos Santos. – São José dos Campos : INPE, 2020.

xx + 133 p. ; (sid.inpe.br/mtc-m21c/2020/04.04.03.50-TDI)

Dissertação (Mestrado em Engenharia e Tecnologia Espaciais/Mecânica Espacial e Controle) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2020.

Orientador : Dr. Marcelo Lopes de Oliveira e Souza.

1. Plataforma MultiMissão. 2. Aviônica Modular Integrada (IMA). 3. IMA Distribuída (DIMA). 4. Concentrador de Dados Remoto (RDC). I.Título.

CDU 629.7.062.2

Aluno (a): *Palmira dos Santos*

Título: "UMA APRECIAÇÃO DO CONCENTRADOR DE DADOS REMOTO (RDC) EM UMA ARQUITETURA IMA DISTRIBUÍDA (DIMA) APLICADO AO CONTROLE DE ATITUDE DA PMM EM MODO NORMAL"

Aprovado (a) pela Banca Examinadora em cumprimento ao requisito exigido para obtenção do Título de *Mestre* em

Engenharia e Tecnologia Espaciais/Mecânica Espacial e Controle

Dr. Mário César Ricci

Presidente / INPE / SJCampos - SP

(X) Participação por Video - Conferência

(X) Aprovado ( ) Reprovado

Dr. Marcelo Lopes de Oliveira e Souza

Orientador(a) / INPE / SJCampos - SP

(X) Participação por Video - Conferência

(X) Aprovado ( ) Reprovado

Dr. Paulo Giácomo Milani

Membro da Banca / INPE / SJCampos - SP

(X) Participação por Video - Conferência

(X) Aprovado ( ) Reprovado

Dr. Fernando José de Oliveira Moreira

Convidado(a) / EMBRAER / SJCampos - SP

(X) Participação por Video - Conferência

(X) Aprovado ( ) Reprovado

Este trabalho foi aprovado por:

( ) maioria simples

(X) unanimidade

São José dos Campos, 15 de abril de 2020

AGRADECIMENTOS

Agradeço ao INPE e à CAPES por oferecerem as condições para a realização deste trabalho.

Aos meus pais, minha irmã Stéphany e meu primo Yago Luiz, pelo amor, carinho, força e incentivo.

Ao meu Orientador, Marcelo Lopes pela paciência, apoio, dedicação, e todo conhecimento compartilhado durante esta caminhada.

Ao meu Professor e Orientador de Graduação, Alexandre Leite, pela força, incentivo e apoio para iniciar o mestrado.

À todos os meus amigos, especialmente Guilherme Mendes e Michel Macena, pela contribuição com suas ideias e sugestões, pelo incentivo, pela força constante em todos os momentos, alegres ou não; e, principalmente, pelo carinho que fez toda a diferença nos momentos de dificuldade.

Ao André Moraes pela confiança, por ter disponibilizado a plataforma para realização deste trabalho e por todo conhecimento compartilhado. Ao Sérgio Penna e ao Miguel Barros que, sempre solícitos, foram essenciais para a conclusão deste trabalho.

Agradeço, também, à Valdirene Moreira por todo suporte, paciência e apoio durante esta caminhada.

RESUMO

Desde os anos 80, a arquitetura federada vem sendo substituída pelas arquiteturas integradas de 1ª e 2ª gerações (IMA e IMA2G), nas áreas aeronáutica, espacial e automobilística, por razões de massa, volume, potência, confiabilidade e redistribuição de funções. Em 2017, no INPE, uma dissertação no Curso ETE/CSE, apreciou, simulou e implementou similarmente a arquitetura IMA2G em um demonstrador por hardware de IMA2G - DIMA. Agora, o presente trabalho apresenta uma apreciação do Concentrador de Dados Remoto (RDC) em uma arquitetura DIMA aplicado ao modo normal da plataforma MultiMissão (PMM). Para fazer isto, esse estudo revê os conceitos básicos e a literatura aberta bem como lista as normas da indústria relacionadas à aplicação dessa arquitetura, e realiza a comunicação do módulo de processamento principal com o RDC no demonstrador, desenvolvido no INPE em 2017, através de 3 implementações. Na 1ª implementação, dividida em 2 casos de teste, a comunicação é realizada com o módulo principal - plataforma Beagle Bone Black BBB1, em modo normal e utiliza o Arduino Uno como RDC. Na 2ª implementação, dividida em 3 casos de teste, a comunicação é realizada com o módulo principal - plataforma *Beagle Bone Black* BBB1, em modo normal e utiliza a plataforma de desenvolvimento *Beagle Bone Black* BBB2 como RDC. Na 3ª implementação, a comunicação é realizada com o módulo principal - plataforma *Beagle Bone Black* BBB1, em modo falhado e utiliza o Arduino Uno como RDC. Os resultados obtidos mostram os efeitos da comunicação em rede através da verificação lógica e análise temporal. Além disso, a 3ª implementação sugere que a arquitetura DIMA é robusta a falhas.

Palavras-chave: Plataforma MultiMissão, Aviônica Modular Integrada (IMA), IMA Distribuída (DIMA), Concentrador de Dados Remoto (RDC).

AN ASSESSMENT OF THE REMOTE DATA CONCENTRATOR IN A DISTRIBUTED INTEGRATED MODULAR AVIONICS (DIMA) APPLIED TO THE NORMAL MODE OF THE MULTIMISSION PLATFORM

ABSTRACT

Since the 1990s, the federated architectures have been replaced by the integrated architectures of 1st and 2nd Generation (IMA and IMA2G), in the aeronautics, space and automotive areas, for reasons of mass, volume, power, reliability and redistribution of functions. In 2017, at INPE, a dissertation in the ETE/CSE Course assessed, simulated and implemented the IMA2G similarly on a demonstrator by hardware of IMA2G – DIMA. Now, this work presents a dissertation about an appreciation of the Remote Data Concentrator (RDC) in a Distributed Integrated Modular Avionics architecture (DIMA) applied to the normal mode of the MultiMission Platform (PMM). To do this it reviews the basic concepts and the open literature as well as lists the industry standards related to the appreciation of this architecture, and performs the communication of the Core Processing Module with the RDC on the demonstrator, developed at INPE in 2017, trough 3 implementations. In the 1st implementation, divided into 2 test cases, the communications is carried out with the core module - Beagle Bone Black BBB1 platform, in normal mode and uses one Arduino Uno as RDC. In the 2nd implementation, divided into 3 test cases, the communication is carried out with the core module - Beagle Bone Black BBB1 platform, in normal mode and uses Beagle Bone Black BBB2 platform as RDC. In the 3rd implementation, the communication is carried out with the core module - Beagle Bone Black BBB1 platform, in failure mode and also uses one Arduino Uno as RDC. The results obtained show the effects of network communication through logical verification and temporal analysis. In addition, the 3rd implementation, suggests that the DIMA architecture is robust to failures.

Keywords: MultiMission Platform, Integrated Modular Avionics (IMA), Distributed IMA (DIMA), Remote Data Concentrator (RDC).

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS E ABREVIATURAS

AIAA	American Institute of Aeronautics and Astronautics
ADCN	Avionics Data Communication Network
ARINC	Aeronautical Radio, Incorporated
CEV	Crew Exploration Vehicle
CPM	Core Processing Module
CPU	Central Process Unit
DHCP	Dynamic Host Configuration Protocol
DIMA	Distributed Integrated Modular Avionics
DME	Distributed Modular Eletronics
DNS	Domain Name System
e.g.	exempli gratia. por exemplo
ESA	European Space Agency
FIFO	Fisrt In First Out
i.e.	Id est. isto é
IEEE	Institute of Electrical and Electronics Engineers
IDE	Integrated Development Environment
IMA	Integrated Modular Avionics
IMA1G	Integrated Modular Avionics First Generation
IMA2G	Integrated Modular Avionics Second Generation
IMA-SP	Integrated Modular Avionics for Space
INPE	Instituto Nacional de Pesquisas Espaciais
IOP	Input/Output Partition
IP	Internet Protocol
IPv6	Internet Protocol version 6
LAN	Local Area Network
LRU	Line Replaceable Unit
MAC	Media Access Control

MMA	Massa Mola Amortecedor
M_p	Maximum overshoot
NASA	National Aeronautics and Space Administration
OSI	Open Systems Interconnection
PMM	Plataforma MultiMissão
REU	Remote Eletronic Unit
RDC	Remote Data Concentrator
RPC	Remote Power Controller
RTCA	Radio Technical Commission for Aeronautics
RTEMS	Real-time Executive for Multiprocessor Systems
SNTP	Simple Network Time Protocol
SWaP	Space, Weight and Power
TCP	Transmission Control Protocol
TFTP	Trivial File Transfer Protocol
TSP	Time and Space Partitioning
UDP	User Datagram Protocol

SUMÁRIO

Pág.

1 INTRODUÇÃO

Este estudo apresenta uma apreciação do concentrador de dados remoto (*Remote Data Concentrator* - RDC) em uma arquitetura aviônica IMA Distribuída (DIMA) aplicado ao controle de atitude da PMM em modo normal.

1.1. Contexto

A arquitetura utilizada por trás dos sistemas aviônicos vem mudando consideravelmente nas últimas décadas, nas áreas aeronáutica, espacial e automobilística. As arquiteturas federadas vêm sendo substituídas pelas arquiteturas integradas de 1ª e 2ª gerações, (IMA e IMA2G/DIMA). Nos anos 90 a indústria aeronáutica, através da sua organização normatizadora *Aeronautical Radio, Incorporated* – ARINC, propôs a arquitetura *Integrated Modular Avionics* (IMA). O sucesso da arquitetura IMA no domínio aeronáutico chamou a atenção da indústria espacial, e tem sido apoiada pelas agências espaciais da Europa (e.g. o projeto IMA-SP, (DINIZ; RUFINO, 2005)), e pela Agência Espacial Americana (NASA) que optou pela arquitetura IMA para o sistema aviônico do veículo de exploração espacial - Orion, (MCCABE; BAGGERMAN; VERMA, 2009). Recentemente, surgiu o conceito *Distributed Integrated Modular Avionics* – DIMA, (FUCHSEN, 2009), que propõe tomar crédito do melhor das arquiteturas Federada e IMA, produzindo a redução das características físicas (massa, volume, potência, conexões, cablagem, estrutura, etc.) e ampliando as características funcionais (confiabilidade, manutenabilidade, disponibilidade, adaptabilidade, reconfigurabilidade, etc.). Assim, essa arquitetura mostra-se ideal para satélites de pequeno e médio porte.

A arquitetura IMA Distribuída, tem como princípio fundamental a separação dos componentes de manipulação de E/S e componentes de processamento de aplicativos, (FUCHSEN, 2009). Uma plataforma típica DIMA é constituída por vários componentes aviônicos, como módulo de processamento principal (*Core Processing Module* - CPM) e concentrador de dados remoto (*Remote Data*

Control - RDC) que são conectados através de um barramento de dados aviônico, (BARROS; NEVES; *et al.*, 2018).

Este trabalho foca um RDC para a DIMA implementada por (MORAES, 2017).

1.2. Objetivo do trabalho de dissertação

O objetivo proposto para este trabalho é realizar uma apreciação do concentrador de dados remoto (RDC) em uma arquitetura IMA distribuída (DIMA) aplicado ao controle de atitude da Plataforma MultiMissão (PMM) em modo normal, ao acrescentar a comunicação por rede e o concentrador de dados remoto ao demonstrador DIMA de (MORAES, 2017). Isto inclui:

a) Realizar o estudo, discussão e seleção do tipo de concentrador de dados remoto (RDC);

b) Realizar a comunicação do módulo principal do demonstrador DIMA de (MORAES, 2017) com o RDC selecionado;

c) Comparar os resultados diretamente do módulo principal (modo normal e modo falhado) com os resultados via RDC após a comunicação.

1.3. Motivação e justificativa

A comunidade de engenharia e da indústria aeroespacial vem se esforçando para aplicar a arquitetura IMA a produtos espaciais. Existem projetos que visam o desenvolvimento do conhecimento necessário à aplicação IMA no espaço patrocinados pela Comunidade Europeia, e.g. IMA-SP, (DINIZ; RUFINO, 2005), e pela Agência Espacial Americana (NASA), que optou pela arquitetura IMA para o sistema aviônico do Módulo de Comando Órion (BAGGERMAN; MCCABE; VERMA, 2009). Além disso, novos métodos e técnicas estão em discussão, gerando conceitos novos denominados IMA2G (IMA segunda geração), (FUCHSEN, 2009).

Dentre esses conceitos, está a aplicação da arquitetura IMA de forma distribuída (DIMA), que propõe tomar crédito: do melhor da arquitetura IMA, i.e., a

otimização dos recursos computacionais disponíveis, a capacidade de reconfiguração e tolerância a falhas, etc; bem como, do melhor na arquitetura federada, i.e., modularidade e a independência entre as funções aviônicas, (MORAES, 2017).

Dadas as características propostas pela arquitetura DIMA, a mesma parece mostrar-se como uma arquitetura ideal para satélites de pequeno e médio porte de alta complexidade voltados a MultiMissão. Dessa forma, conclui-se que os novos projetos espaciais, em médio prazo, tendem a implementar essas tecnologias que estão sendo desenvolvidas agora.

Isso entusiasma o acréscimo da comunicação por rede e de um concentrador de dados remoto ao demonstrador DIMA de (MORAES, 2017), que implementa um sistema de controle de Atitude (SCA) compatível com os satélites em desenvolvimento no INPE. Note-se que uma plataforma típica DIMA é constituída por vários componentes aviônicos, e.g., módulo de processamento principal (*Core Processing Module* - CPM) e concentrador de dados remoto (*Remote Data Control* - RDC); que são conectados através de um barramento de dados aviônico, (BARROS; NEVES; *et al.*, 2018).

1.4. Organização deste trabalho

Este trabalho está organizado da seguinte forma:

O Capítulo 1 - Introdução, apresenta uma breve introdução às arquiteturas IMA, IMA2G contextualizando o seu desenvolvimento e lista resumidamente algumas motivações para a aplicação a sistemas espaciais. Também é apresentado nesse capítulo o objetivo e a organização do trabalho.

O Capítulo 2 - Conceitos básicos e revisão da literatura, apresenta os conceitos sobre a arquitetura IMA, IMA2G, DIMA e algumas considerações sobre suas aplicações espaciais. Apresenta também os conceitos sobre o concentrador de dados remoto e suas formas de realização, o conceito de sistemas em tempo

real e sistema operacional em tempo real; e, por fim, um breve comentário sobre tópicos relevantes para a comunicação em rede realizada neste trabalho.

O Capítulo 3 - Formulação do problema e abordagens para sua solução, define uma abordagem de estudo a partir do problema apresentado, propondo uma solução para esse problema ao acrescentar a comunicação por rede e um concentrador de dados remoto.

O Capítulo 4 - Implementações, descreve as implementações propostas nesta pesquisa.

O Capítulo 5 - Resultados e análises, apresenta as análises dos resultados obtidos das implementações realizadas.

O Capítulo 6 - Dificuldades encontradas, apresenta algumas das dificuldades encontradas ao longo desta pesquisa, assim como alguns comentários sobre as mesmas.

O Capítulo 7 - Comentários finais, conclusões, e sugestões para trabalhos futuros, tece os comentários e as considerações finais sobre a pesquisa, bem como propõe sugestões de próximas pesquisas.

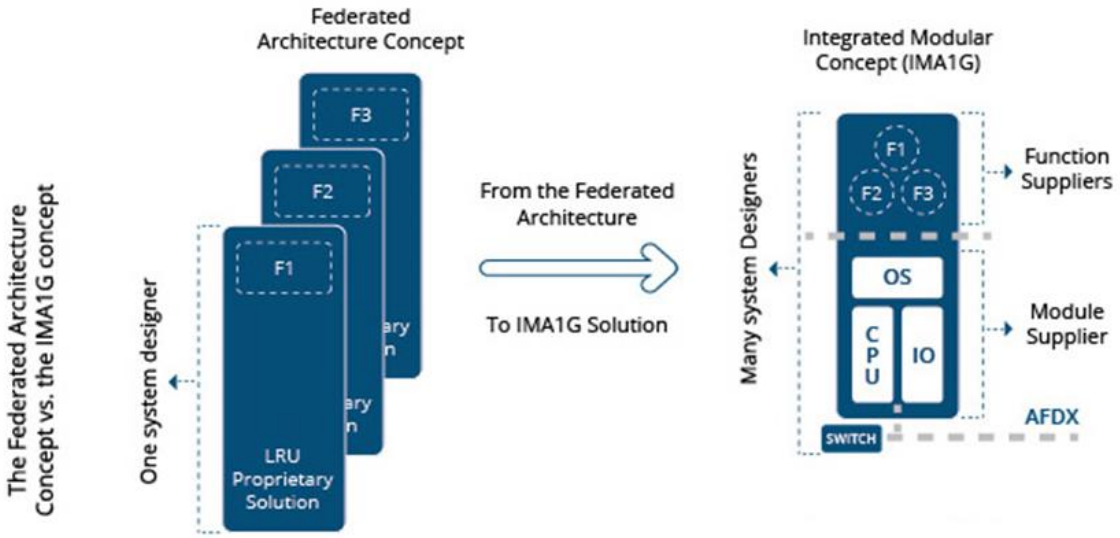
2 CONCEITOS BÁSICOS E REVISÃO DA LITERATURA

2.1. Conceituação sobre a arquitetura IMA

Na arquitetura federada dos sistemas aviônicos, cada subsistema é implementado em uma LRU (Line Replaceable Unit), que é uma unidade autocontida que possui todos os recursos necessários para execução da sua função, (WATKINS; WALTER, 2007). A arquitetura federada é consagrada como uma solução robusta e confiável, bastante aplicada na indústria aeronáutica desde a sua origem e continuamente usada no momento presente na indústria espacial, (MORAES, 2017). Contudo, discutida desde os anos 1980, a limitação da arquitetura federada tornou-se evidente nos anos 2000 com a aumento do número de sistemas eletrônicos embarcados, dado que, para cada novo subsistema há a necessidade da integração de uma nova LRU com suas demandas de volume, massa e potência elétrica (*Space, Weight and Power* – SWaP), conectores, cablagem, etc. Com o objetivo de atender essa demanda crescente de recursos, a indústria aeronáutica, na década de 90, através da sua organização ARINC, propôs a arquitetura IMA (Integrated Modular Avionics), que é capaz de integrar várias funções aviônicas na mesma plataforma computacional, (PRISAZNUK, 1992).

De acordo com (MORAES, 2017), a arquitetura IMA pode ser assimilada como "uma virtualização da arquitetura federada, onde as LRUs são substituídas por funções de software independentes entre si e que compartilham os recursos computacionais disponíveis". A Figura 2.1 apresenta o conceito da arquitetura federada *versus* a arquitetura IMA1G.

Figura 2.1 - Arquitetura Federada x Arquitetura IMA1G.

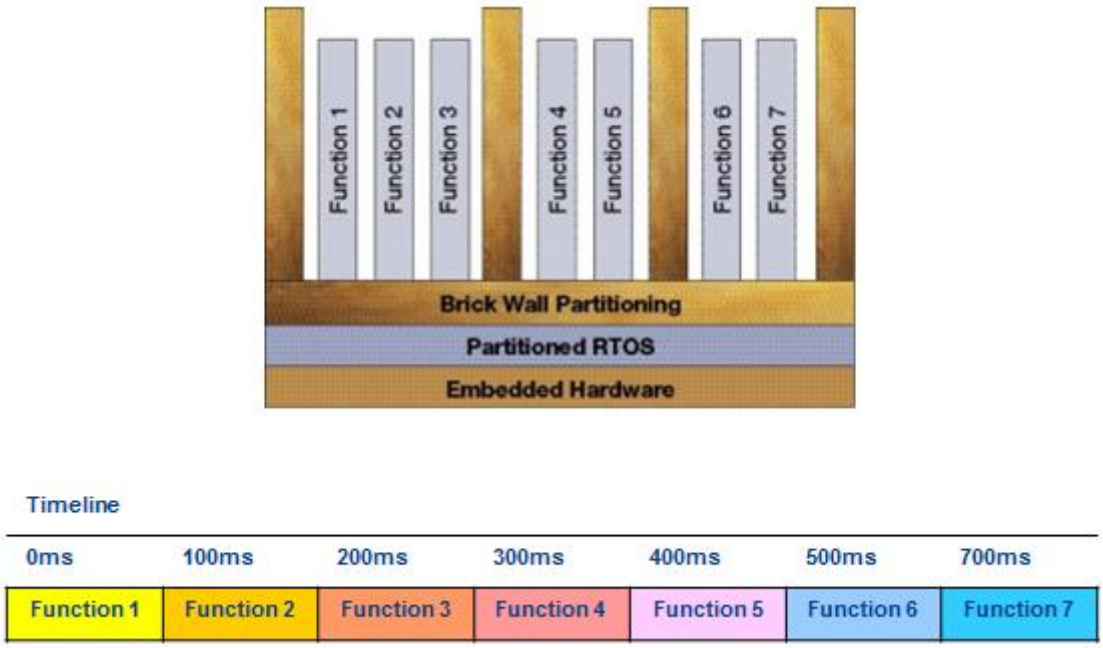


Fonte: Maret (2015).

A separação física existente entre as LRU's, na arquitetura IMA é fornecida virtualmente para as funções em execução na mesma CPU, em que o desempenho de cada função não é afetado por outras. Essa separação é fornecida através do particionamento comum dos recursos e a atribuição desses recursos particionados à função, (RUFINO, 2009).

A característica fundamental da IMA é garantir que o compartilhamento dos recursos seja realizado de forma que uma função não interfira na outra, mesmo com a presença de falhas em alguma das funções, (MORAES, 2017). Dessa forma, o sistema operacional IMA atua de modo que determinada função tenha acesso aos recursos apenas em determinada: janela de tempo, parte da memória, entradas e saídas. Assim, o sistema operacional através do "*Brick Wall Partitioning*", garante que falhas ocorridas em determinada função não interfira em outras; isso preserva a integridade do sistema (MORAES, 2017), Figura 2.2.

Figura 2.2 - *Brick Wall Partitioning.*



Fonte: Moraes (2017).

Os conceitos da arquitetura IMA desenvolvidos pela indústria aeroespacial por meio da *Aeronautical Radio Incorporated* (ARINC) / *Airlines Electronic Engineering Committee* (AEEC) já estão sedimentados e são utilizados na indústria aeronáutica. As bases de implementação da arquitetura são definidas pela ARINC e RTCA, elas também fornecem os *guidelines* para a certificação dos sistemas novos, listados na Tabela 2.1

Tabela 2.1 - Normas ARINC e RTCA para IMA.

NORMA	TÍTULO
ARINC SPECIFICATION 650	Integrated Modular Avionics Packaging and Interfaces
ARINC SPECIFICATION 650-1	Design Guidance for Integrated Modular Avionics
ARINC SPECIFICATION 652	Guidance for Avionics Software Management

continua

Tabela 2.1 – Conclusão.

ARINC SPECIFICATION 653 P0-1	Avionics Application Software Standard Interface Part 0, Overview of Arinc 653
ARINC SPECIFICATION 653 P1-4	Avionics Application Software Standard Interface Part 1, Required Services
ARINC SPECIFICATION 653 P2-3	Avionics Application Software Standard Interface Part 2, Extended Services
ARINC SPECIFICATION 653 P3a	Avionics Application Software Standard Interface, Part 3a, Conformity Test Specification for Arinc 653 Required Services
ARINC SPECIFICATION 653 P4	Avionics Application Software Standard Interface Part 4, Subset Services
ARINC SPECIFICATION 653 P5	Avionics Application Software Standard Interface Part 5, Core Software Recommended Capabilities
ARINC SPECIFICATION 654	Environmental Design Guidelines for Integrated Modular Avionics Packaging and Interfaces
ARINC SPECIFICATION 664P1-1	Aircraft Data Network Part 1, Systems Concepts and Overview
ARINC SPECIFICATION 664P2-2	Aircraft Data Network Part 2, Ethernet Physical and Data Link Layer Specification
ARINC SPECIFICATION 664P3-2	Aircraft Data Network Part 3, Internet-Based Protocols and Services
ARINC SPECIFICATION 664P4-2	Aircraft Data Network Part 4, Internet-Based Address Structure & Assigned Numbers
ARINC SPECIFICATION 664P5	Aircraft Data Network Part 5, Network Domain Characteristics and Interconnection
ARINC SPECIFICATION 664P7-1	Aircraft Data Network Part 7, Avionics Full Duplex Switched Ethernet (AFDX) Network
ARINC SPECIFICATION 664P8-1	Aircraft Data Network Part 8, Interoperation With Non-IP Protocols and Services
RTCADO-297	Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations

Fonte: Produção do Autor.

Sobre a transição da arquitetura federada para a arquitetura IMA é preciso fazer algumas considerações, (WATKINS; WALTER, 2007). A Tabela 2.2 apresenta comparações entre as arquiteturas. Nesse contexto questiona-se sobre em que momento e escopo a migração de uma arquitetura para outra é vantajosa, (WATKINS; WALTER, 2007).

Tabela 2.2 - Arquitetura Federada x Arquitetura IMA.

Arquitetura Federada	**Arquitetura IMA**
Utilização na Indústria	
Amplamente usada nas indústrias aeronáutica e espacial.	Usada na indústria aeronáutica. Nova na indústria espacial.
Integração e certificação	
Integração relativamente fácil; Certificação relativamente menos complexa.	Integração e certificação muito mais complexa, demandando processos de desenvolvimento diferenciados.
Demanda de SWaP	
Cada subsistema tem sua demanda de SWaP; Existe limite econômico e técnico ao número de subsistemas que podem ser implemetados.	Menor demanda de SWaP comparado ao mesmo número de subsistemas da arquitetura federada.
Utilização de recursos	
Inerente subutilização de recursos.	Proporciona compartilhamento de recursos, viabilizando número de funcionalidades muito maior, considerando o recurso computacional disponível.

Fonte: Produção do Autor.

2.2. Evolução da arquitetura IMA

Nos últimos anos, uma nova geração da arquitetura IMA tem sido discutida e propostas de otimização estão sendo consideradas. Essas propostas de otimização ficaram conhecidas como IMA2G. A demanda imposta pelo mercado aeronáutico por novos serviços de transporte aéreo estimula essa evolução, (FUCHSEN, 2009).

A evolução da arquitetura IMA foi debatida no projeto SCARLETT (*Scalable & Reconfigurable Electronics Platforms and Tools*), cf. (FUCHSEN, 2009). As propostas de otimização consideradas e apresentadas em (FUCHSEN, 2009) são:

a) O projeto SCARLETT propõe o conceito Eletrônica Modula Distribuída (*Distributed Modular Eletronics* - DME), também conhecido como DIMA (*Distributed* IMA) que introduz a separação física entre as funções de gerenciamento de I/O e o processamento das funções aviônicas;

b) O uso do *design* de processadores *multicore* para suportar a evolução da performance computacional. Essa estratégia dita a necessidade de garantir determinismo no processamento em tempo real para funcionalidades "*safety critical*". Para isso, tem se trabalhado em métodos e técnicas que viabilizam o desenvolvimento e a certificação desses sistemas com o uso de processadores *multicore*;

c) Uma das principais inovações proposta é a implementação de mecanismos avançados de reconfiguração através do gerenciamento de redundância e alocação dos recursos dinamicamente em nível de plataforma em vez de estaticamente em nível de sistema;

d) Reduzir custos de desenvolvimento e manutenção através da disponibilidade de uma cadeia de ferramentas consistentes para todo processo de desenvolvimento, certificação, integração e manutenção; dada as características de complexidade e de alta integração dos seus componentes da arquitetura IMA;

e) Capacitar a plataforma IMA2G para integrar e executar aplicações desenvolvidas em sistemas operacionais diferentes, através da virtualização e generalização de hardware. Dessa forma um software desenvolvido em uma plataforma computacional federada pode ser virtualizado no contexto IMA.

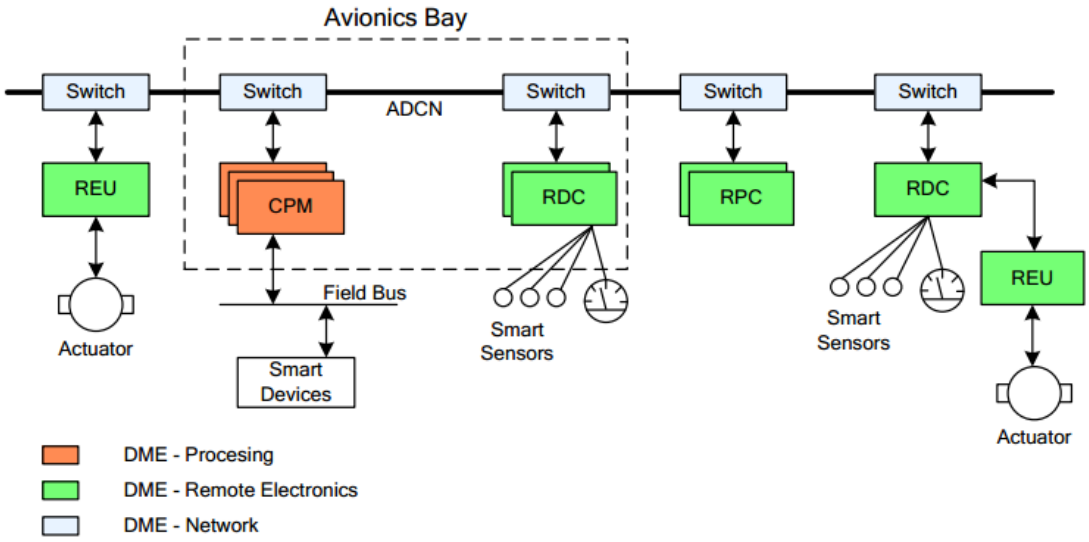
> A arquitetura IMA2G se propõe a aproveitar o melhor da arquitetura federada, i.e., modularidade e a independência entre as funções aviônicas, e o melhor da arquitetura IMA, i.e., a otimização dos recursos computacionais disponíveis, a capacidade de reconfiguração e tolerância a falhas, etc. (MORAES, 2017, p. 37).

Como material de referência, destaca-se (WILKINSON, 2005), (JAKOVLJEVIC; ADEMAJ, 2013), (JAKOLJEVIC; ADEMAJ, 2015) e (GASKA; WATKIN; CHEN, 2015).

2.3. Eletrônica modular distribuída – Distributed IMA (DIMA)

A arquitetura DME, também conhecida como IMA Distribuída, tem como princípio fundamental a separação dos componentes de manipulação de E/S e componentes de processamento de aplicativos. Eles se comunicam através de uma rede de comunicação de dados aviônicos (*Avionics Data Communication Network* – ADCN). Para cumprir as metas IMA2G, outro fundamento básico é o suporte de reconfiguração do *hardware* e *software* de todos os módulos da plataforma durante a operação, (FUCHSEN, 2009). A Figura 2.3 apresenta a arquitetura DME.

Figura 2.3 - Arquitetura DME.



Fonte: Fuchsen (2009).

Essa arquitetura é composta por, (FUCHSEN, 2009):

a) Módulos de processamento (*Core Processing Module* - CPM): que hospedam aplicativos aviônicos de criticalidade;

b) Um barramento de dados determinístico e de alto desempenho (*Advanced Deterministic Communication Network* - ADCN);

c) Concentradores de dados remoto (*Remote Data Concentrator* - RDC): que fazem as interfaces entre os CPM's e os atuadores e sensores;

d) Unidades eletrônicas remotas (*Remote Eletronic Unit* - REU): que são dispositivos dedicados a uma função específica e precisam suportar os princípios de reconfiguração da DME. Elas podem estar ligadas direto ao ADCN ou ligada a um RDC através de um barramento de dados dedicado;

e) Controlador de potência remoto (*Remote Power Controller* - RPC): que é responsável pela distribuição de energia entre os componentes do sistema aviônico;

f) *Switch* ARINC 664: que gerencia toda a configuração de rede e é responsável por assegurar a comunicação determinística.

2.4. Aplicação da arquitetura IMA no contexto espacial

Conforme apresentado em (WINDSOR; HJORTNAES, 2009) e (WINDSOR; DEREDEMPT; DE-FERLUC, 2011), a indústria espacial compartilha das mesmas necessidades básicas da indústria aeronáutica, como: a redução de demandas de SWaP, a capacidade de alto nível de integração de sistemas, além de ferramentas, métodos e técnicas para gerenciar um número maior de funções aviônicas embarcadas.

A avaliação dos prós e contras da arquitetura IMA, ao iniciar um projeto, é essencial para escolher uma solução mais adequada. Existem projetos que visam o desenvolvimento do conhecimento necessário à aplicação da IMA no espaço. Destacam-se:

No projeto patrocinado pela Agência Espacial Americana (NASA) são feitas considerações sobre a escolha do sistema aviônico para os veículos do Programa *Constellation*, descrito em (MCCABE; BAGGERMAN; VERMA, 2009). Nesse projeto, diferentes abordagens de arquitetura aviônica foram escolhidas para dois dos principais veículos do Programa *Constellation:* Veículo de Exploração Orion – *Orion Crew Exploration Vehicle* (CEV) e o Veículo de Lançamento Ares I. Diferenças consideráveis de restrição de projeto e de requisitos operacionais existem entre os veículos Orion e Ares I. Isso levou à escolha de arquiteturas aviônicas diferentes. A arquitetura federada para o projeto do Ares I foi a opção mais interessante, dada a não exigência de desenvolvimento de novas tecnologias, que seria preciso para as interfaces IMA com os sistemas legados. Para o projeto do Orion, a arquitetura IMA foi escolhida devido à necessidade de proteção contra falhas e à flexibilidade de operação exigida pelo projeto, cf. (MCCABE; BAGGERMAN; VERMA, 2009).

O projeto IMA-SP patrocinado pela Comunidade Europeia, teve como objetivo a avaliação da aplicabilidade da arquitetura integrada modular a sistemas espaciais, tendo como premissa as diferenças entre os sistemas aeronáuticos e espaciais, e o uso do hardware aviônico espacial já utilizado. Nessa pesquisa, identificaram-se as tecnologias que precisavam ser desenvolvidas para o uso da

arquitetura IMA nos sistemas espaciais, cf. (WINDSOR; HJORTNAES, 2009), (WINDSOR; DEREDEMPT; DE-FERLUC, 2011).

No Brasil, existe o projeto IMA4Brazil desenvolvido com a cooperação entre a empresa brasileira Equatorial, a empresa GMV em Portugal e o Instituto Tecnológico de Aeronáutica (ITA). Esse projeto teve como objetivo validar a aplicação da arquitetura IMA a plataformas espaciais, cf. (HOLANDA, 2014).

No INPE, existem trabalhos voltados para a aplicabilidade da arquitetura IMA no espaço, são eles: a dissertação de mestrado de (TAGAWA, 2013), que realizou o estudo de um controlador de atitude em um simulador de aviônica modular integrada-SIMA, e a dissertação de mestrado de (MORAES, 2017), que realizou uma apreciação, simulação e implementação da arquitetura IMA distribuída e a sua aplicação a sistemas espaciais. Esses trabalhos utilizaram como caso de estudo um sistema de controle de atitude (SCA) compatível com os dos satélites em desenvolvimento no INPE.

Nota-se então um esforço da comunidade de engenharia e da indústria aeroespacial em aplicar a arquitetura IMA a produtos espaciais.

2.5. Concentrador de dados remoto

Uma plataforma típica DIMA é constituída por vários componentes aviônicos, como o módulo de processamento principal (*Core Processing Module* - CPM) e o concentrador de dados remoto (*Remote Data Control* - RDC) que são conectados através de um barramento de dados aviônico, (BARROS; NEVES; *et al.*, 2018).

O RDC é um dispositivo específico que reúne os dados de diferentes sensores/atuadores e os envia para o módulo principal através de uma rede IMA, (SILVA, 2012).

Na plataforma DIMA, os RDC's são introduzidos para distribuir E/S através de uma aeronave, (BARROS; NEVES; *et al.*, 2018).

O RDC possui uma interface de E/S para conexão a um ou mais dispositivos de E/S, e uma interface de rede para conexão a um módulo principal. Ele permite a

comunicação entre o(s) dispositivo(s) de E/S e o módulo principal, e também pode possuir um conjunto de instruções para conduzir autonomamente um disposto de saída ligado à sua interface E/S, em um evento de perda do comando original do módulo principal. Isso evita a necessidade de um sistema de controle secundário redundante conectando o dispositivo de saída ao módulo principal, o que reduz a complexidade e reduz consideravelmente o peso, (TODD; NITSCHE, 2015).

O RDC pode ainda avaliar a validade do comando original e conduzir o dispositivo de saída de acordo com os melhores recursos de comando, dando maior confiabilidade, detecção de falhas e redundância (TODD; NITSCHE, 2015).

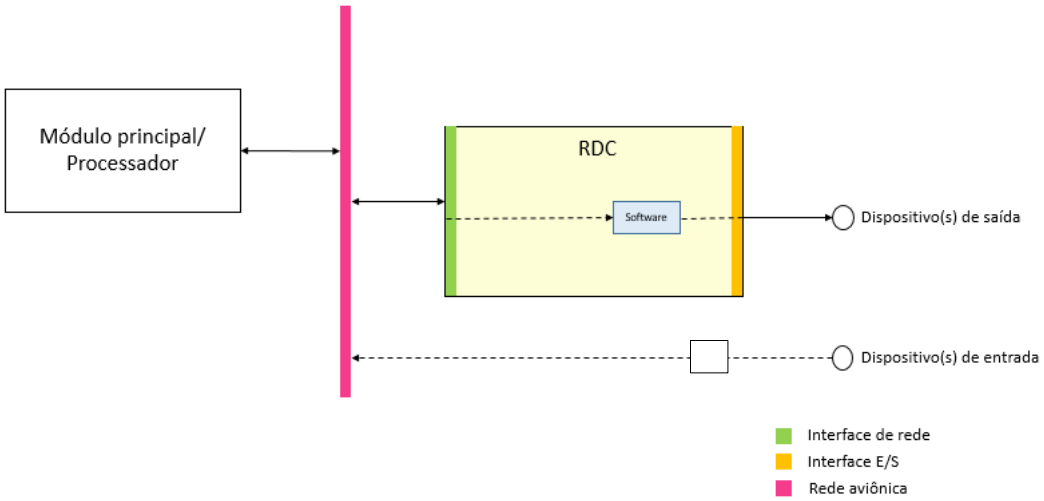
2.5.1. Formas de realização do RDC

De acordo com (TODD; NITSCHE, 2015), há quatro formas de realização do RDC:

2.5.1.1. Forma de realização 1

A primeira e mais simples consiste em uma rede aviônica incluindo um RDC que tem uma interface E/S para conexão com um ou mais dispositivos de E/S, e uma interface para conexão com o módulo principal, como mostra a Figura 2.4.

Figura 2.4 - Forma de realização 1 do RDC.



Fonte: Adaptado de Todd e Nitsche (2015).

Nela: a interface de rede do RDC está conectada ao módulo principal processador, e.g., controlador, através da rede aviônica que possui dispositivo de E/S; o dispositivo de saída, e.g., atuador, está conectado à rede via RDC; enquanto que o dispositivo de entrada, e.g., sensor, está ligado por meios alternativos, como, por exemplo, outro RDC.

Em condições normais de operação, o dispositivo de saída, e.g., atuador, recebe um sinal de comando gerado pelo módulo principal, e.g., controlador, em resposta ao sinal do dispositivo de entrada, e.g., sensor. Esse comando é transmitido através da rede para a interface de rede do RDC e para o software.

O módulo do software pode ter operadores lógicos condicionais tal que julgue se um comando válido, para o dispositivo de saída, foi recebido do módulo principal; e, então, envia esse comando para o dispositivo de saída.

Porém, no caso do RDC perder o comando oriundo do módulo principal, o *software*, que tem armazenado estados padrões predeterminados para o dispositivo de saída, não recebe um comando válido; então, na ausência de um

comando válido oriundo do módulo principal, o RDC assume o controle do dispositivo de saída (como no Controle Digital Direto, usado em indústrias).

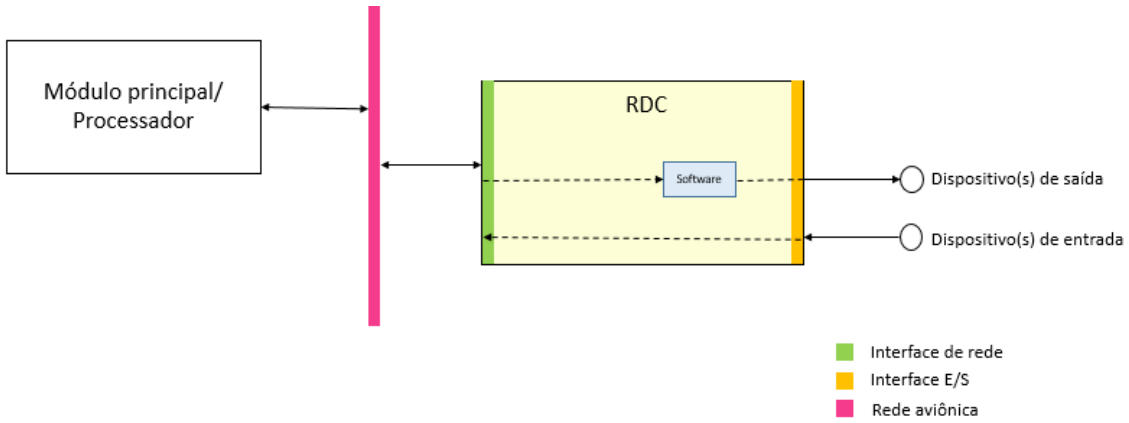
Como o RDC pode estar conectado a vários dispositivos de saída, o software deve armazenar vários estados padrões predeterminados para cada um dos dispositivos de saída. Desse modo, o RDC pode autonomamente conduzir uma variedade de dispositivos de saída caso o comando principal seja perdido, (TODD; NITSCHE, 2015).

2.5.1.2. Forma de realização 2

Na segunda forma de realização, Figura 2.5, o RDC tem os dispositivos de entrada e os de saída conectados à sua interface E/S. Não importa se o dispositivo de entrada e o dispositivo de saída estão conectados ao mesmo RDC desde que não haja nenhuma conexão interna.

O modo como o *software* opera para controlar os dispositivos de saída é exatamente o mesmo que na forma de realização descrita anteriormente.

Figura 2.5 - Forma de realização 2 do RDC.

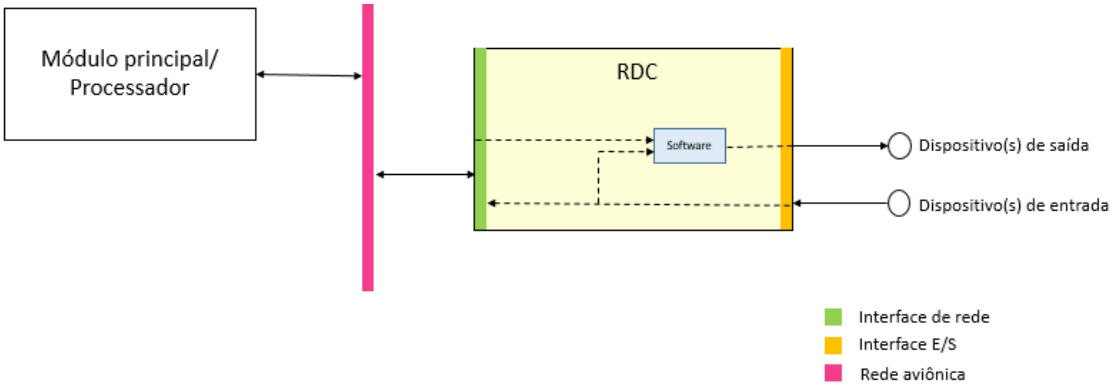


Fonte: Adaptado de Todd e Nitsche (2015).

2.5.1.3. Forma de realização 3

A terceira forma de realização inclui uma versão mais complexa do RDC. O RDC possui um *software* que inclui lógica combinatória e passa a receber também o sinal do dispositivo de entrada, ou seja, o *software* recebe dois sinais, o comando para o dispositivo de saída e o sinal do dispositivo de entrada, Figura 2.6.

Figura 2.6 - Forma de realização 3 do RDC.



Fonte: Adaptado de Todd e Nitsche (2015).

O *software* guarda um conjunto de instruções para converter o sinal do dispositivo de entrada em um comando local. Esse conjunto de instruções deve ser similar ao usado pelo módulo principal para gerar o comando principal.

Durante a operação normal da rede aviônica, o *software* é configurado para comparar o comando principal, gerado pelo módulo principal, com o comando local e, então, enviar ao dispositivo de saída o comando principal se ele e o comando local estiverem em acordo. Essa configuração permite a verificação e *crosschecking* do sinal de comando para melhorar segurança e reduzir a probabilidade de uma operação errônea.

O *software* pode, alternativamente ou adicionalmente, ser configurado para enviar ao dispositivo de saída o comando local no lugar do comando principal. Essa configuração é útil quando a validade do comando principal é duvidosa e a

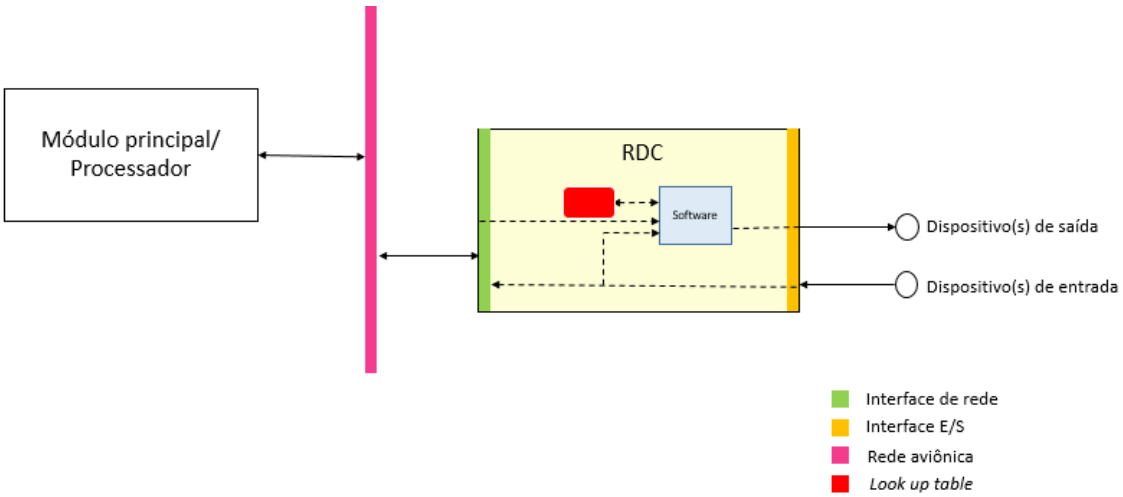
convicção é insuficiente para a função de saída ser realizada de maneira segura baseada no comando principal. O sinal de comando local é prioritário nesse caso e é utilizado no dispositivo de saída.

O *software* pode alternativamente, ou adicionalmente, ser configurado para o caso de perda do comando vindo do módulo principal, então o *software* entrega a interface de E/S um estado predeterminado para assim o dispositivo de saída ir para uma posição predeterminada, (TODD; NITSCHE, 2015).

2.5.1.4. Forma de realização 4

A quarta opção é um RDC que inclui um *software* com uma *Lookup table*, Figura 2.7. Na *Lookup table* é armazenado um conjunto de comandos locais para serem enviados ao dispositivo de saída dependendo do estado do comando principal e/ou sinal de entrada. O *software* não armazena o conjunto de comandos locais; então, ele contém uma quantidade menor de instruções e é mais fácil de se testar. Armazenando os comandos principais na *Lookup table*, um número maior de estados predeterminados para o dispositivo de saída pode ser facilmente fornecido.

Figura 2.7 - Forma de realização 4 do RDC.



Fonte: Adaptado de Todd e Nitsche (2015).

O *software* acessa a *Lookup table* e seleciona um dos vários estados predeterminados para o dispositivo de saída, de acordo com o comando principal e/ou o comando local. Tendo selecionado o estado predeterminado apropriado, o RDC conduz o dispositivo de saída ao estado predeterminado. O RDC não pode fornecer um completo nível de controle para o dispositivo de saída como seria fornecido pelo módulo principal, mas um nível maior de granularidade no controle do dispositivo de saída pode ser conseguido se comparado as outras formas de realização do RDC.

Então, o RDC pode fornecer o controle autonomamente de um ou mais dispositivos de saída conectados a ele no caso de uma invalidade, ou perda total do comando principal. Esse tipo de controle elimina a necessidade de um sistema de controle secundário para controlar o dispositivo de saída no caso de perda do comando principal.

Essa opção também torna possível fornecer um RDC com funcionalidade para ter um controle completamente autônomo, dentro dos limites da *Lookup table*, sobre um ou mais dispositivos de saída conectados a ele sem a necessidade de qualquer comando principal vindo do módulo principal em condições de operação normal, (TODD; NITSCHE, 2015).

2.6. Sistema de tempo real

Aplicações com requisitos de tempo real estão cada vez mais comuns, elas variam em relação à complexidade e às necessidades de garantia no atendimento de restrições temporais. Aplicações que apresentam a característica de estarem sujeitas a restrições temporais, mais rigorosas (*Hard Real-Time*) ou menos rigorosas (*Soft Real-Time*), são agrupadas nos sistemas de tempo real, (FARINES; FRAGA; OLIVEIRA, 2000).

No entanto, ainda se cometem erros no momento de definir se o sistema é ou não de tempo real. A maioria dos sistemas de tempo real são projetados e implementados a partir da visão de que tempo real quer dizer somente melhoria no desempenho, que é uma visão errada, (STANKOVIC, 1988). Além disso é

comum a interpretação equivocada de um sistema de tempo real como um sistema que efetua os cálculos computacionais de forma rápida. De acordo com (FARINES; FRAGA; OLIVEIRA, 2000, p. 4), “Um Sistema de Tempo Real (STR) é um sistema computacional que deve reagir a estímulos oriundos do seu ambiente em prazos específicos”.

Apenas a questão temporal não é suficiente para a implementação correta de um sistema de tempo real; o sistema de tempo real precisa entregar o resultado mais correto cumprindo o prazo específico. Segundo (FARINES; FRAGA; OLIVEIRA, 2000, p. 4), “O comportamento correto de um sistema de tempo real, não depende só da integridade dos resultados obtidos (correção lógica ou ‘*correctness*’) mas também dos valores de tempo em que são produzidos (correção temporal ou ‘*timeliness*’) ”.

Mais importante que a rapidez de cálculo é o conceito de previsibilidade. Então durante o desenvolvimento de um sistema de tempo real, é necessário o uso de ferramentas e metodologias adequadas que permitam verificar o comportamento e a implementação do mesmo como previsíveis, (FARINES; FRAGA; OLIVEIRA, 2000).

> Um sistema de tempo real é dito ser previsível (“predictable”) no domínio lógico e no temporal quando, independentemente de variações ocorrendo a nível de hardware (i.e., desvios do relógio), da carga e de falhas, o comportamento do sistema pode ser antecipado, antes de sua execução, (FARINES; FRAGA; OLIVEIRA, 2000, p. 5).

A partir do ponto de vista da segurança, os sistemas de tempo real podem ser classificados em: sistemas de tempo real brando, *soft real time systems* e sistemas de tempo real duro, *hard real time systems*, (FARINES; FRAGA; OLIVEIRA, 2000).

2.6.1. Agendamento de tempo real

O ponto central na previsibilidade do comportamento de sistemas de tempo real é formado pelas técnicas e conceitos de agendamento, (FARINES; FRAGA; OLIVEIRA, 2000). Separado em partes, cada parte do sistema é uma tarefa que

tem suas características definidas, o agendamento é que define o momento em que cada tarefa deve ser executada.

Toda tarefa está sujeita a prazos, seus *deadlines*. Quando uma tarefa é concluída após o seu *deadline,* ela pode causar falhas catastróficas ou, no mínimo implicar em uma diminuição do desempenho do sistema; isso depende se a tarefa é crítica ou não. As tarefas também são classificadas em tarefas periódicas e aperiódicas. Nas tarefas periódicas o instante de tempo no qual ela chega é conhecido, e a sua chegada ocorre em intervalos de tempo constantes. Nas tarefas aperiódicas, não se conhece o instante de tempo no qual ela chega, podendo então a sua chegada ocorrer de maneira aleatória, (FARINES; FRAGA; OLIVEIRA, 2000).

2.7. Sistema operacional de tempo real

Sistema operacional de tempo real é uma categoria dos sistemas operacionais, ele é desenvolvido para atender as características temporais dos programas que nele são executados.

2.7.1. AIR

O AIR é um sistema operacional em tempo real particionado em tempo e espaço, desenvolvido pela empresa GMV em uma parceria do meio acadêmico com a indústria aeroespacial. Ele é baseado no RTEMS, é *open source* e é compatível com o padrão ARINC 653.

Uma aplicação AIR é um conjunto do *Kernel* AIR, dos executáveis da partição e das configurações. As partições do AIR são *software* que são executados a comando do AIR. Elas definem as aplicações exatas do sistema em execução, no local em que o AIR está sendo executado. As partições não são fornecidas pelo AIR, a criação delas é de responsabilidade do usuário, (GMV INNOVATING SOLUTIONS, 2018).

2.7.1.1. Comunicação entre partições

A comunicação entre as partições é um mecanismo oferecido pelo sistema operacional da GMV. A comunicação é conduzida através de mensagens e pode ser realizada entre duas ou mais partições. (GMV INNOVATING SOLUTIONS, 2018). Uma mensagem é definida “como um bloco contíguo de dados de comprimento finito”, (GMV INNOVATING SOLUTIONS, 2018, p. 84).

A mensagem é enviada de uma única partição e pode ser recebida em uma ou mais partições. O canal é o mecanismo utilizado para vincular as partições, que define um *link* lógico entre elas. As partições acessam os canais através das portas, que fornecem os recursos necessários para uma partição ler ou gravar uma mensagem no canal, (GMV INNOVATING SOLUTIONS, 2018).

Dois tipos de portas são implementadas no sistema operacional da GMV, *queuing* e *sampling.* Nas portas *queuing* os valores são enfileirados por gravadores e desenfileirados por leitores, semelhante a um *buffer* FIFO. Uma porta *sampling* é um espaço de memória atualizado a uma determinada taxa, (GMV INNOVATING SOLUTIONS, 2018).

2.8. Comunicação em rede

O modelo OSI (*Open Systems Interconnection model*), lançado em 1984 pela Organização Internacional para a Normatização (*International Organization for Standardization*-ISO), tem como objetivo estruturar e padronizar o mundo da comunicação e da rede de dados.

O modelo OSI consiste em sete camadas funcionais, cada camada contém um número definido de funções. As camadas desse modelo são: 7) aplicação; 6) apresentação; 5) sessão; 4) transporte; 3) rede; 2) enlace de dados e 1) física.

Atualmente, a tecnologia utilizada em redes locais (LAN) predominante é a Ethernet, que opera na 2) camada de enlace de dados e na 1) camada física.

Ethernet é um protocolo de conexão entre redes, considerado o padrão mais utilizado em redes locais. O protocolo Ethernet foi padronizado pelo IEEE como 802.3, (802.3-2018 - IEEE Standard for Ethernet, 2018). Ele tem se tornado uma

escolha frequente como meio físico de transmissão de dados, devido à sua disponibilidade de componentes padronizados, e pela sua alta velocidade de transmissão.

O protocolo *Transmission Control Protocol* (TCP), (POSTEL, 1980) e o protocolo *User Datagram Protocol* (UDP), (POSTEL, 1980) são protocolos que operam na 3) camada de transporte do modelo OSI. Basicamente, a diferença entre eles é que o TCP garante a entrega de todos os pacotes de dados entre o emissor e o receptor, o que o torna mais confiável, enquanto que o UDP é mais simples e não garante essa entrega de pacote de dados. Se o pacote se perder na comunicação, normalmente, com o protocolo UDP não existe uma solicitação de reenvio do mesmo.

3 FORMULAÇÃO DO PROBLEMA E ABORDAGENS PARA SUA SOLUÇÃO

O problema proposto é fazer uma apreciação do concentrador de dados remoto (RDC) em uma arquitetura IMA distribuída aplicado ao controle de atitude da PMM em Modo Normal. Isto pretende demonstrar a aplicabilidade da arquitetura IMA Distribuída (DIMA) à sistemas espaciais e tecer considerações sobre a separação entre componentes de processamento de aplicativos e componentes de manipulação de I/O.

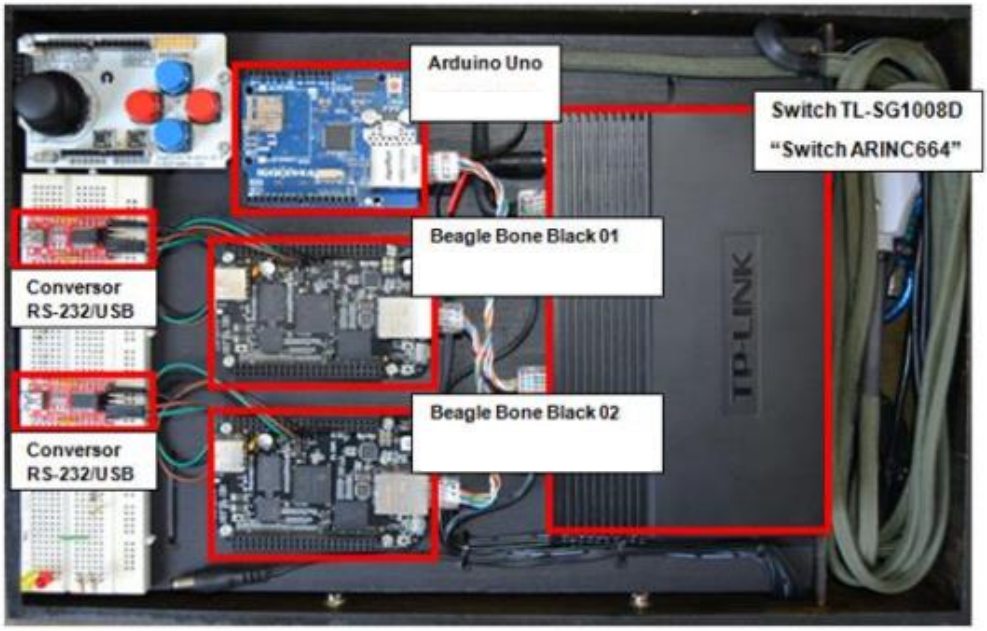
A demonstração é realizada por meio do estudo de caso do demonstrador DIMA de (MORAES, 2017). O demonstrador DIMA de (MORAES, 2017) implementa em uma arquitetura DIMA um sistema de controle e Atitude (SCA) compatível com os satélites em desenvolvimento no INPE. Este trabalho acrescenta a comunicação por rede e um concentrador de dados remoto ao demonstrador DIMA e analisa como este acréscimo o afeta.

3.1. Demonstrador DIMA

O demonstrador de (MORAES, 2017) representa um sistema aviônico na arquitetura IMA Distribuída. Ele é constituído por: duas plataformas de desenvolvimento *Beagle Bone Black*, um Arduino Uno, um *switch* Ethernet e dois conversores RS-232/USB, Figura 3.1.

O demonstrador implementa um modelo do sistema Massa Mola Amortecedor (MMA) e um modelo compatível com o Sistema de Controle de Atitude (SCA) da PMM em uma plataforma de desenvolvimento *Beagle Bone Black,* que representa um computador de bordo com o sistema operacional AIR. Ambos os modelos são executados simultaneamente e compartilham a plataforma computacional no contexto da arquitetura IMA.

Figura 3.1 - Demonstrador DIMA.

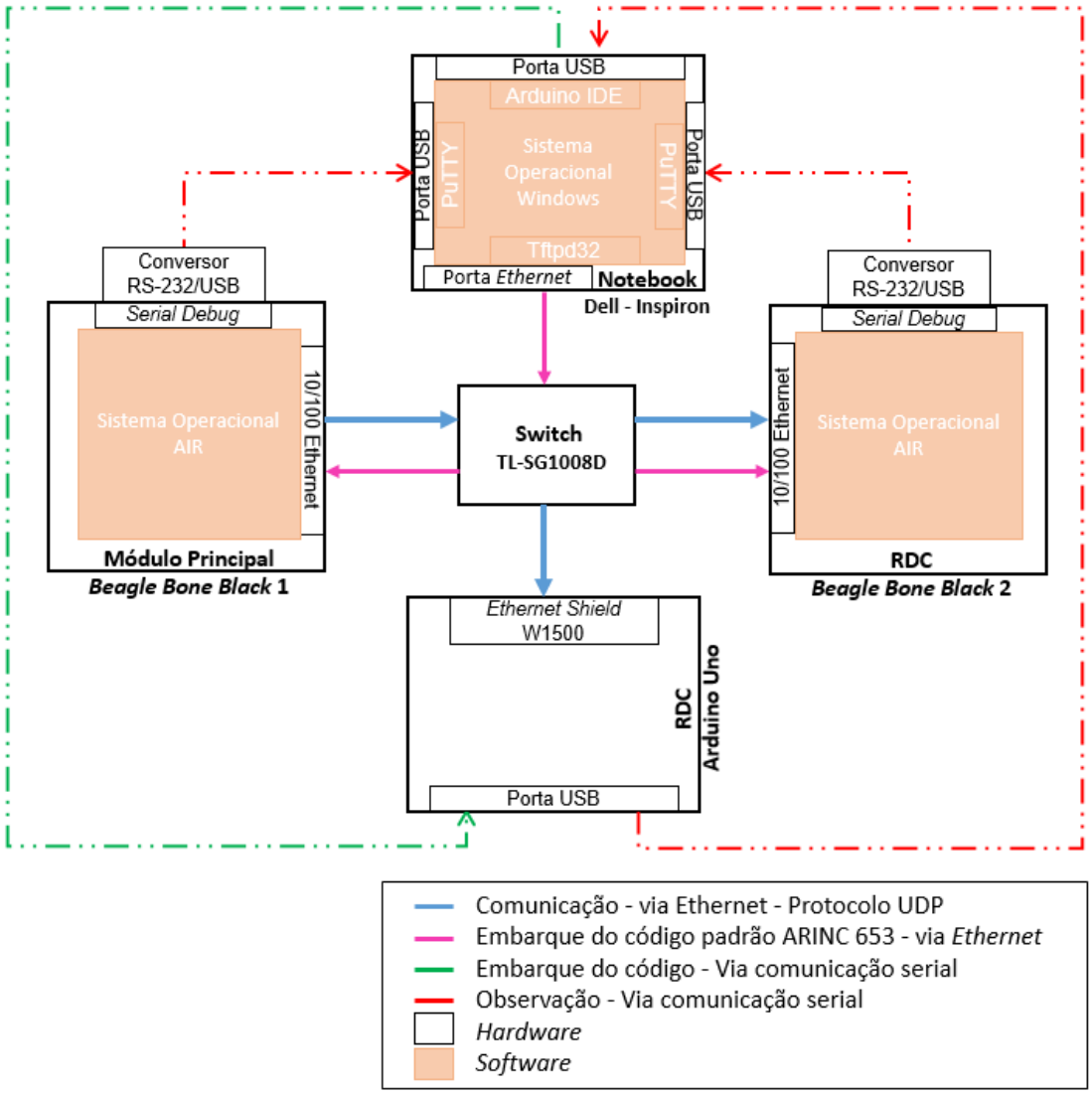


Fonte: Adaptado de Moraes (2017).

O Sistema Operacional AIR fornecido pela empresa GMV, é um sistema operacional *open source* compatível com o padrão ARINC 653, desenvolvido em parceria do meio acadêmico com a indústria aeroespacial, (MORAES, 2017), motivada pelo interesse da ESA em aplicar os conceitos de TSP (*Time and Space Partitioning*) à missões espaciais, cf. (RUFFINO; CRAVEIRO; *et al.*, 2011).

As placas de desenvolvimento *Beagle Bone Black, e o* Arduino Uno se comunicam via rede Ethernet e utilizam o protocolo UDP como protocolo de comunicação. A rede Ethernet no demonstrador representa uma rede padrão ARINC 664. A Figura 3.2 apresenta o esquema de comunicação do demonstrador DIMA .

Figura 3.2 - Esquema de comunicação do demonstrador DIMA.



Fonte: Produção do Autor.

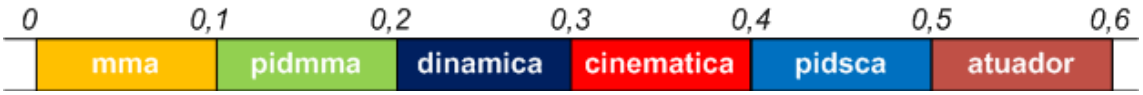
3.1.1. Modos de operação

O demonstrador DIMA de (MORAES, 2017) utilizado neste trabalho possui dois modos de operação: o modo normal e o modo com falha:

3.1.1.1. Modo normal

Na lógica do sistema aviônico desenvolvida em (MORAES, 2017), o modo normal implementa um modelo do sistema Massa Mola Amortecedor (MMA) e um modelo compatível com o Sistema de Controle de Atitude (SCA) da PMM. O sistema MMA é composto por duas partições: “mma” e “pidmma”, e o sistema SCA é composto por seis partições: “dinâmica”, “cinemática”, “pid” e “atuador”. Cada partição é executada por 0.1s com período de 0.6s. As partições são executadas conforme o agendamento apresentado na Figura 3.3.

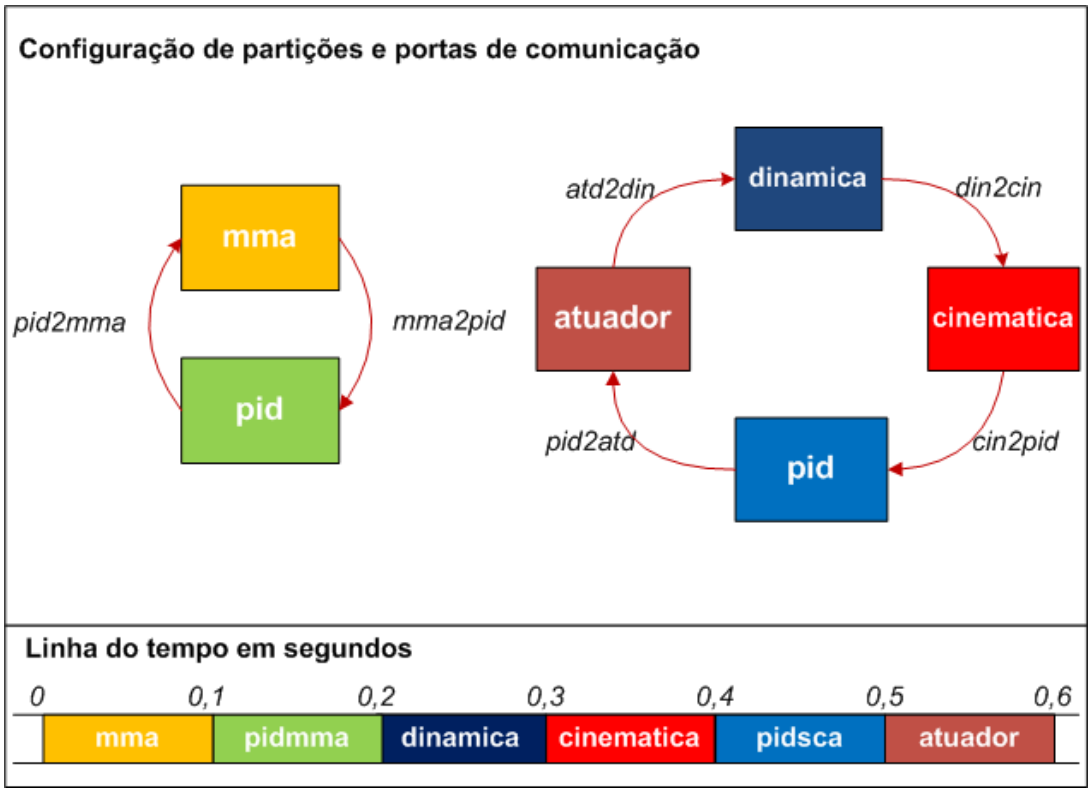
Figura 3.3 - Agendamento do modo normal.



Fonte: Moraes (2017).

A Figura 3.4 apresenta a configuração das partições e das portas de comunicação utilizadas no modo normal.

Figura 3.4 - Configuração de partições e portas de comunicação.



Fonte: Moraes (2017).

3.1.1.2. Modo com falha

Baseado no modo normal, no modo com falha é inserido um erro de computação no sistema SCA para que se comprove que os demais sistemas e/ou partições continuam funcionando e que o sistema é capaz de se reconfigurar tratando a condição de falha inserida. Então o demonstrador é capaz de mostrar que: se houver uma falha em uma das partições as demais continuam operando ("*fault tolerance*"); tem a capacidade de detectar falhas e prevenir que a falha prejudique a operação do sistema ("*fault prevention*") através de reconfiguração; é capaz de detectar e remover falhas ("*fault correction*"), (MORAES, 2017).

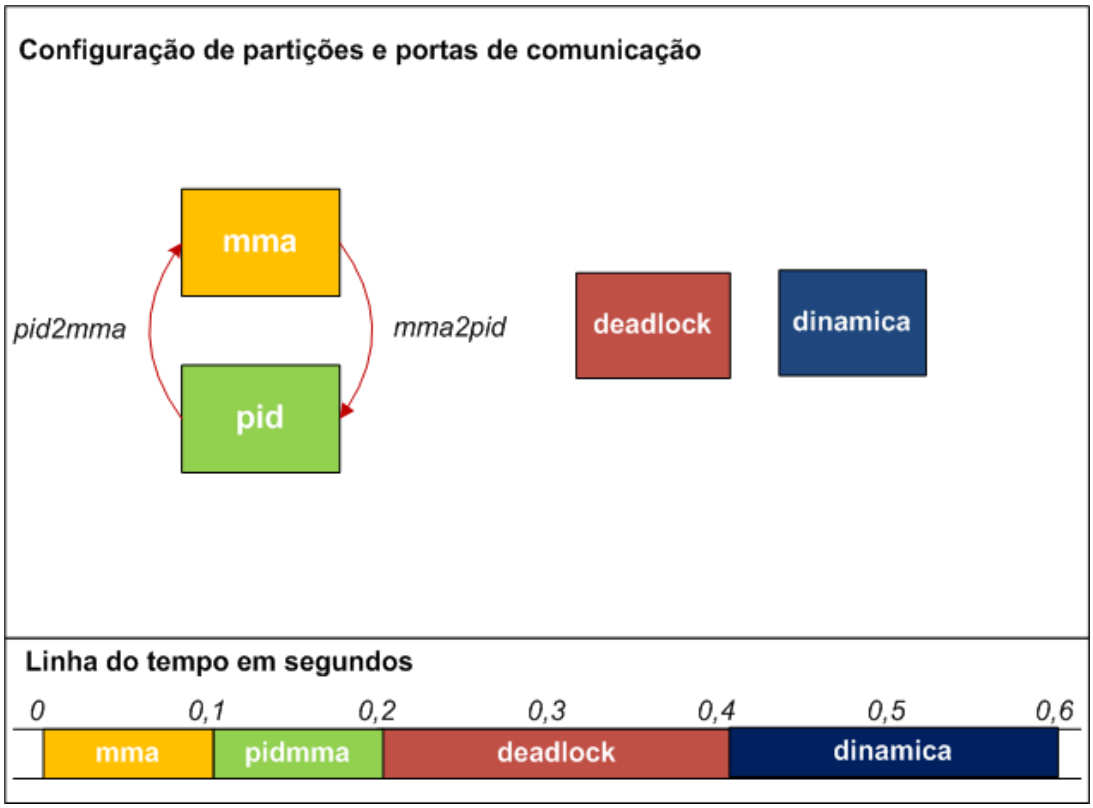
O erro de computação é inserido no código de uma das partições do sistema SCA e a função "*Health Monitoring*" do AIR é usada para reconfigurar a partição e isolar a falha, (MORAES, 2017).

O erro de computação se trata de um acesso a endereço inválido de memória, requerido através do endereçamento de um ponteiro nulo.

> O erro ("*error*" – i.e., endereçamento de memória errado) resultante desse código gera uma falha ("*fail*"- i.e., tentativa de acesso a um endereço de memória inválido) que é isolada pelo "*Health Monitoring"* do AIR e uma reconfiguração ocorre carregando um segundo "schedule" e levando o sistema a operar em modo falha, ou seja, outra configuração de módulos é executada em uma temporização diferente e o sistema SCA segue operando em modo falha. (MORAES, 2017, p. 147).

A função realiza a reconfiguração por meio do descarte do agendamento do modo normal e carregamento em seguida de um segundo agendamento específico para tratar a falha. O novo agendamento ativado pela rotina de "*Health Monitoring*" continua executando o sistema SCA e executa as partições "*deadlock*" e "dinâmica". Então, nessa condição o sistema MMA continua a ser executado normalmente e no sistema SCA está ativo apenas o contador para reinicialização do sistema, Figura 3.5. Após cinco ciclos, o sistema é reiniciado e retorna à execução normal, cf. (MORAES, 2017).

Figura 3.5 - Configuração de partições e portas de comunicação - Modo de Falha.



Fonte: Moraes (2017).

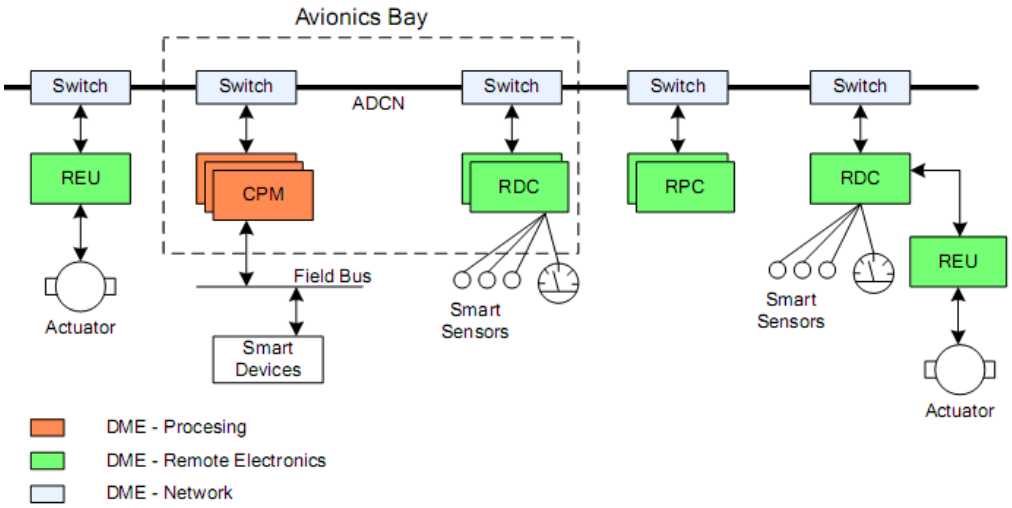
A lógica do sistema aviônico é implementada nas partições IMA através de código C que é compilado no ambiente AIR, gerando uma imagem, i.e., um código executável padrão ARINC 653. O carregamento do código ARINC 653 na plataforma de desenvolvimento *Beagle Bone Black,* é realizado através da interface Ethernet da placa com um *notebook*. Para isso, é utilizado o aplicativo gratuito e de código livre Tftpd32, que cria um servidor padrão IPv6 capaz de suportar comunicações DHCP, TFTP, DNS e SNTP, cf. (MORAES, 2017).

3.2. O problema no contexto do demonstrador

É necessário notar que no demonstrador DIMA de (MORAES, 2017), todo o sistema encontra-se implementado em um único lugar (módulo principal - BBB1), i.e., controlador, planta e dispositivos de E/S. Em um sistema real, na arquitetura IMA Distribuída, os componentes encontram-se separados; os dispositivos de

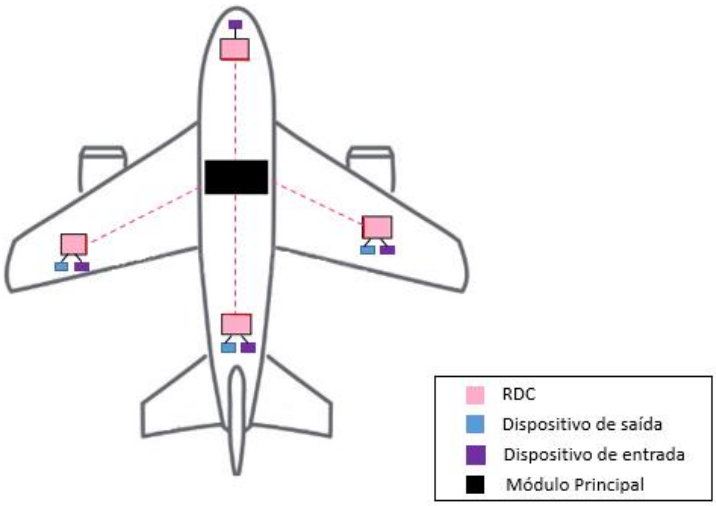
E/S são conectados, diretamente ou indiretamente, ao RDC; o RDC se comunica com o módulo de processamento principal através de uma rede aviônica, como apresentado na Figura 3.6, na Figura 3.7 e na Figura 3.8.

Figura 3.6 - Arquitetura DME.



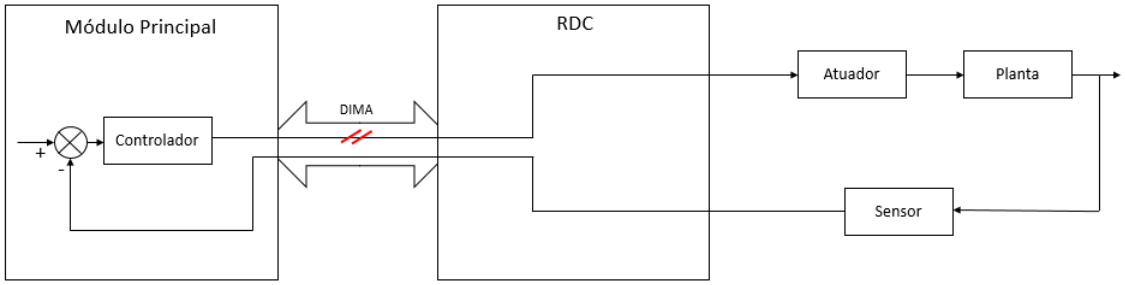
Fonte: Fuchsen (2009).

Figura 3.7 - Arquitetura DIMA.



Fonte: Produção do Autor.

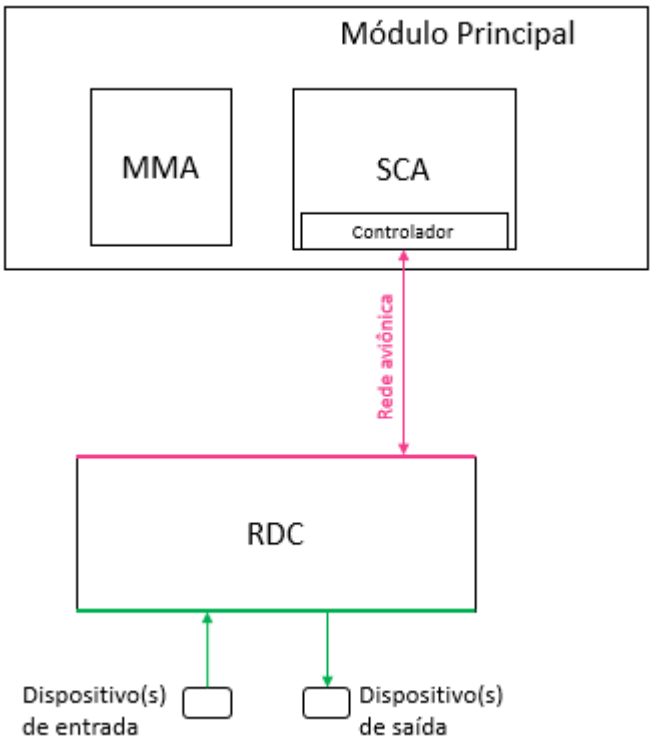
Figura 3.8 - Sistema de controle em uma arquitetura DIMA.



Fonte: Produção do Autor.

Idealmente a arquitetura do demonstrador DIMA é apresentada na Figura 3.9, onde a interface de rede do RDC está conectada ao módulo principal através da rede aviônica que possui dispositivo E/S, conectados à rede via RDC.

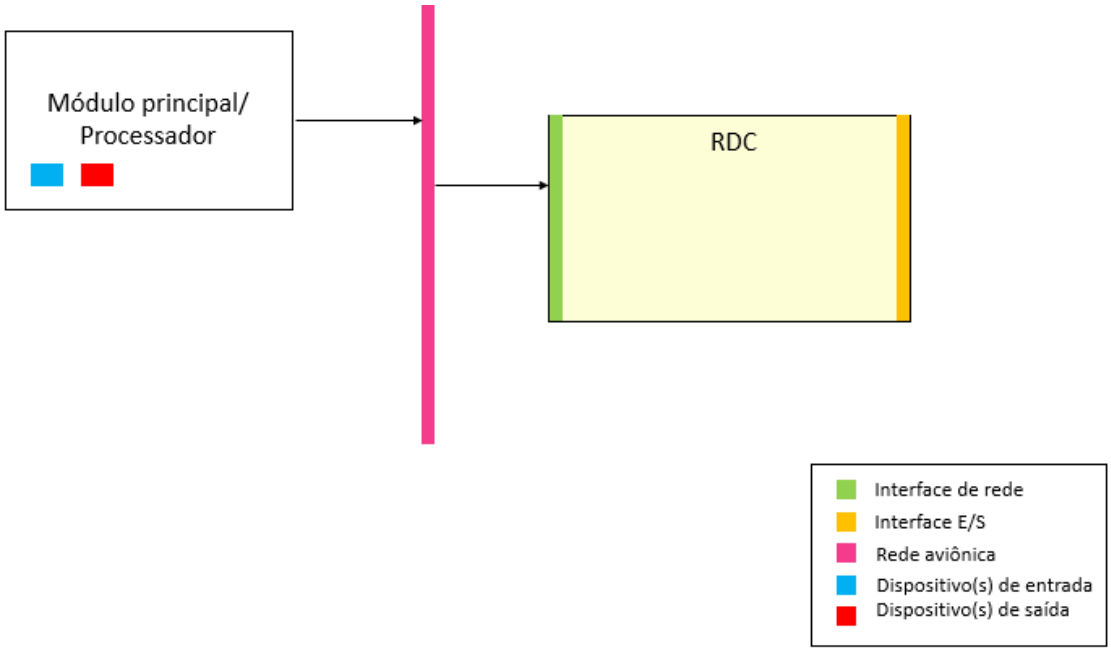
Figura 3.9 - Arquitetura ideal do demonstrador DIMA.



Fonte: Produção do Autor.

No demonstrador DIMA, os dispositivos de saída e de entrada estão implementados dentro do módulo principal. Por isso, baseado nas formas de realização do RDC apresentadas no Capítulo 2, e na realidade do demonstrador DIMA, uma nova forma de realização do RDC é considerada, apresentada na Figura 3.10.

Figura 3.10 - Forma de realização 0 do RDC.



Fonte: Produção do Autor.

Na forma de realização 0, a malha do sistema não é fechada pelo RDC. Nessa forma de realização, a malha está fechada dentro do módulo principal. A alteração de todo o sistema do demonstrador DIMA de forma a ajustá-lo para uma arquitetura típica DIMA demanda um grande esforço e estudo, além de não ser trivial. Porém a forma de realização 0 é rica e instrutiva, pois, através da mesma é possível demonstrar diretamente os efeitos de uma falha de comunicação, e.g., perda de dados, e é possível demonstrar indiretamente os efeitos de uma falha de comunicação, ao inserir essa falha na malha fechada dentro do módulo principal.

As implementações realizadas neste trabalho, no demonstrador DIMA que utiliza a forma de realização 0 do RDC, visam analisar a comunicação do módulo principal com o RDC e o efeitos de possíveis perturbações da mesma no sistema. A análise das possíveis alterações dos dados enviados pela rede Ethernet, é feita por meio da comparação entre o sinal oriundo do módulo principal e o sinal recebido pelo RDC. Os dados enviados e a análise realizada diferem em cada implementação de acordo com o objetivo da mesma.

4 IMPLEMENTAÇÕES

As implementações deste trabalho são realizadas no demonstrador DIMA, que representa um sistema aviônico na arquitetura IMA Distribuída, constituído por duas plataformas de desenvolvimento *Beagle Bone Black*, BBB1 e BBB2, e um Arduino Uno. Esses dispositivos se comunicam via rede Ethernet e utilizam o protocolo UDP como protocolo de comunicação. A rede Ethernet no demonstrador representa uma rede padrão ARINC 664.

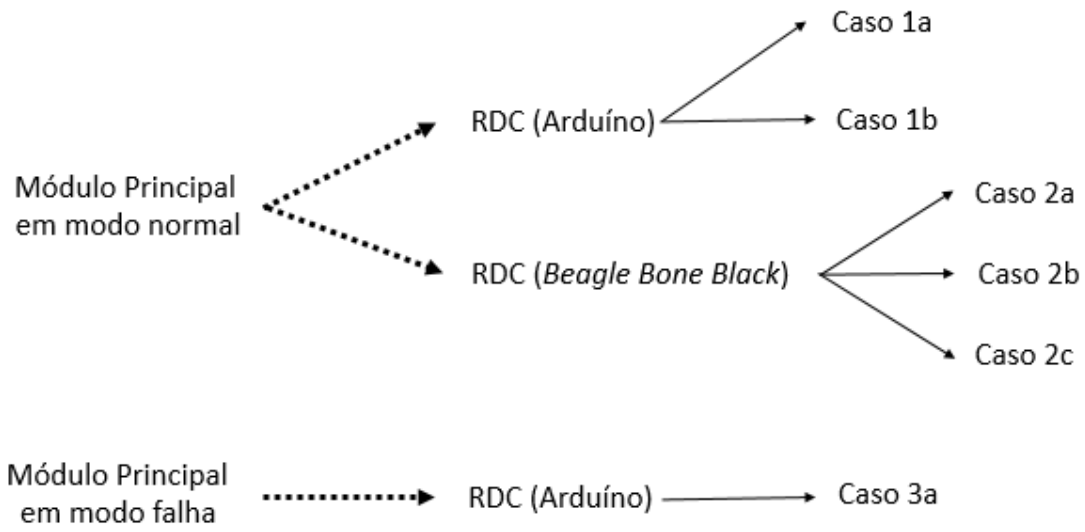
No demonstrador DIMA, o módulo principal é uma plataforma *Beagle Bone Black* BBB1 que possui os sistemas SCA e MMA implementados. Os dispositivos de saída e de entrada, no demonstrador DIMA, estão implementados dentro do módulo principal. Além disso, o módulo principal possui duas possibilidades de modo de operação: modo normal e modo com falha.

O RDC, pode ser implementado na plataforma Arduino ou em uma segunda plataforma *Beagle Bone Black* BBB2*.* Essas possibilidades de implementação têm como objetivo explorar como cada plataforma reage e interfere na comunicação entre o módulo principal e o RDC.

Então este trabalho analisa as possíveis alterações dos dados enviados pelo sistema de comunicação (rede Ethernet), que ocorrem durante a comunicação do módulo principal com o RDC, pelo seguinte Projeto de Experimentos:

A árvore de possibilidades da Figura 4.1 e a Tabela 4.1 a seguir apresentam as possibilidades de implementação realizadas neste trabalho.

Figura 4.1 - Árvore de possibilidades de implementação.



Fonte: Produção do Autor.

Tabela 4.1 - Implementações.

Implementação	**Módulo Principal**	**RDC**	**Objetivo**	**Dado enviado**
1.a	BBB1	Arduino	Envio do sinal de controle e análise do efeito da comunicação sobre o mesmo.	u (x), u (y), u (z).
1.b	BBB1	Arduino	Análise temporal do envio e do recebimento dos dados.	Número de pulsos do relógio decorridos desde que o sistema é inicializado.
2.a	BBB1	BBB2	Análise temporal do envio e do recebimento dos dados; Frequência de recebimento > Frequência de envio.	Número de pulsos do relógio decorridos desde que o sistema é inicializado.

continua

Tabela 4.1 – Conclusão.

2.b	BBB1	BBB2	Análise temporal do envio e do recebimento dos dados; Frequência de recebimento = Frequência de envio.	Número de pulsos do relógio decorridos desde que o sistema é inicializado.
2.c	BBB1	BBB2	Análise temporal do envio e do recebimento dos dados. Frequência de recebimento < Frequência de envio.	Número de pulsos do relógio decorridos desde que o sistema é inicializado.
3.a	BBB1 (Modo falha)	Arduino	Comprovar o isolamento das partições diante de falhas numa delas.	Sinal de controle do Sistema MMA.

Fonte: Produção do Autor.

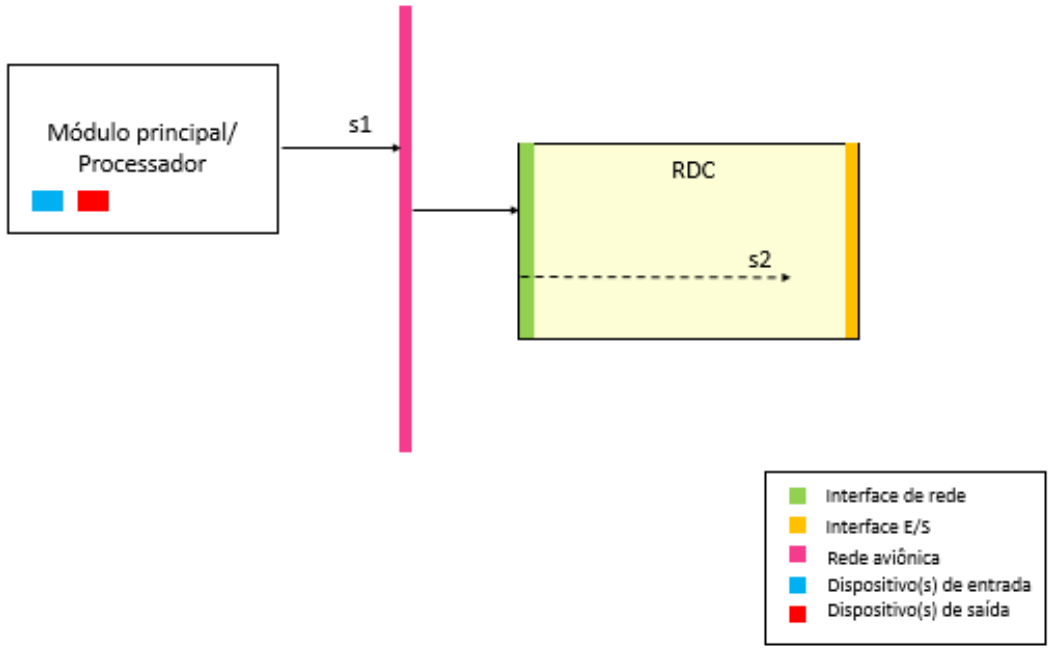
A lógica das implementações do módulo principal deste trabalho é programada em código C e possui também arquivos de configuração em XML. Os códigos são compilados no ambiente AIR, o qual gera um executável padrão ARINC 653. O carregamento do código ARINC 653 na plataforma de desenvolvimento *Beagle Bone Black* BBB1, é realizado através da interface Ethernet da placa com um notebook, por meio do aplicativo servidor Tftpd32.

A lógica das implementações do RDC para a plataforma de desenvolvimento *Beagle Bone Black* BBB2*,* é a mesma do módulo principal. A lógica das implementações do RDC para a plataforma Arduino, é programada e embarcada através do ambiente de desenvolvimento integrado, i.e., Arduino IDE.

4.1. Módulo principal no modo normal

No Modo Normal de operação, o módulo principal do demonstrador DIMA se comunica com o RDC conforme apresentado na Figura 4.2, que demonstra o fluxo de sinal da rede, em que *s1* é o sinal oriundo do módulo principal, e *s2* é o sinal recebido pelo RDC.

Figura 4.2 - Módulo principal e RDC.

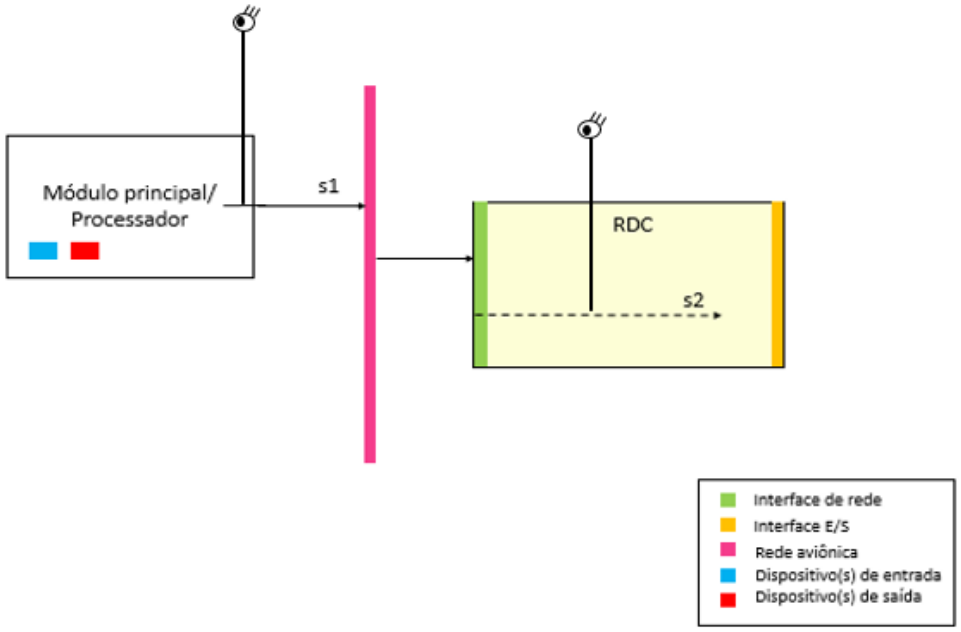


Fonte: Produção do Autor.

A análise das possíveis alterações dos dados enviados pela rede Ethernet, é feita por meio da comparação entre o sinal enviado, *s1*, e o sinal recebido, *s2*. O sinal enviado, *s1*, é observado através da interface de comunicação serial RS232 do *Beagle Bone Black* BBB1, gerenciada pelo software terminal serial PuTTY. A observação do sinal recebido, *s2*, depende da plataforma utilizada para a implementação. Quando o RDC é implementado no *Beagle Bone Black* BBB2, o sinal é observado do mesmo modo que o sinal enviado, *s1*. De outro modo, quando implementado no Arduino, o sinal recebido, *s2,* é observado

através do monitor serial do Arduino. As observações dos sinais estão representadas pelo “ícone do olho” na Figura 4.3 a seguir.

Figura 4.3 - Observação do sinal entre o módulo principal e o RDC.



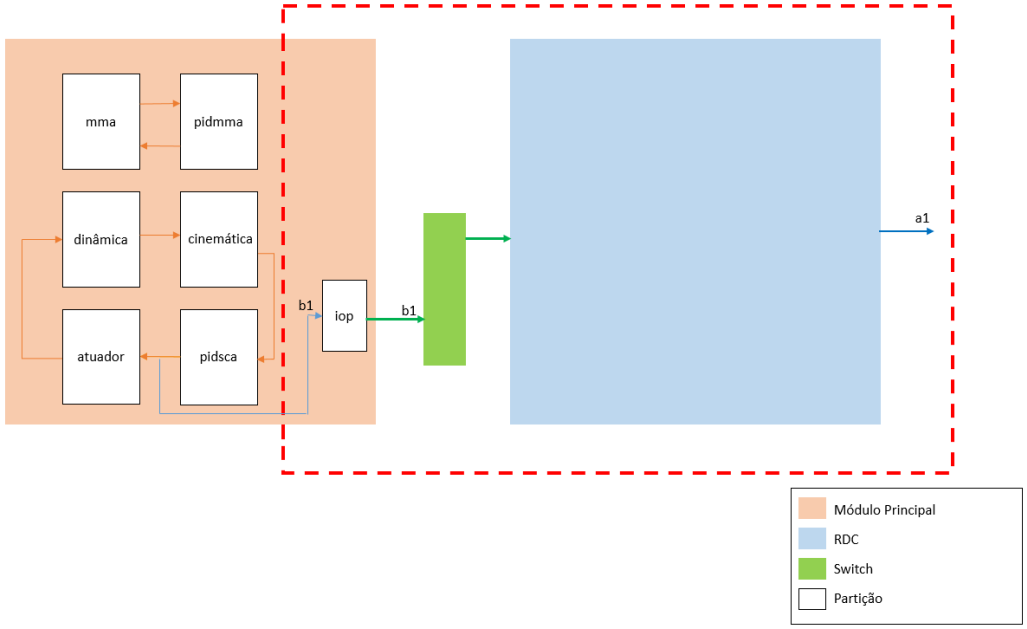
Fonte: Produção do Autor.

O módulo principal, no trabalho de (MORAES, 2017) é composto por 6 partições: mma, pidmma, dinâmica, cinemática, atuador e pidsca; essas partições são executadas no sistema operacional AIR durante 0.1s com período de 0.6s. Para que haja a comunicação entre o RDC e o módulo principal, via rede Ethernet, é necessário criar uma nova partição (IOP1) no módulo principal, Figura 4.4. A partição IOP1 apenas gerencia essa comunicação. Observa-se que, o sinal *b1* recebido por essa partição é o mesmo sinal que é enviado à rede Ethernet.

A nova partição, IOP1, é executada no sistema operacional AIR durante 0.1s. Ao adicioná-la o período de todas as partições passa a ser de 0.7s. O passo de integração dos sistemas SCA e MMA definido por (MORAES, 2017) é de 0.6s. Para execução de todas as partições, incluindo a IOP1, é estimado o passo de integração em 0.7s.

Na Figura 4.4 é possível observar, na área delimitada pelo quadrado vermelho tracejado, os acréscimos ao sistema DIMA do trabalho do André Moraes, (MORAES, 2017): o RDC e a comunicação via rede Ethernet.

Figura 4.4 - Comunicação entre o módulo principal e o RDC.



Fonte: Produção do Autor.

Os dados escolhidos para envio ao RDC são os dados oriundos da partição "*pidsca*" do modelo compatível com o sistema de controle de atitude (SCA) da PMM, que tem como função controlar a posição do satélite em relação a um sistema de coordenadas de referência, cuja a origem é o seu centro de massa. Essa partição é escolhida pois o sinal gerado pela mesma é o sinal de controle, que tem como objetivo fazer um sistema, uma planta, se comportar ou ter um desempenho de acordo com o desejado, de acordo com as especificações. Além disso, esse sinal é sensível aos efeitos da comunicação em rede, pois ao ser enviado em rede, esse sinal pode ser perdido, corrompido e/ou chegar atrasado. Tais efeitos podem impedir o sistema de atingir o desempenho desejado, i.e., a planta pode instabilizar, ter falhas ou falência. Portanto, a análise das possíveis

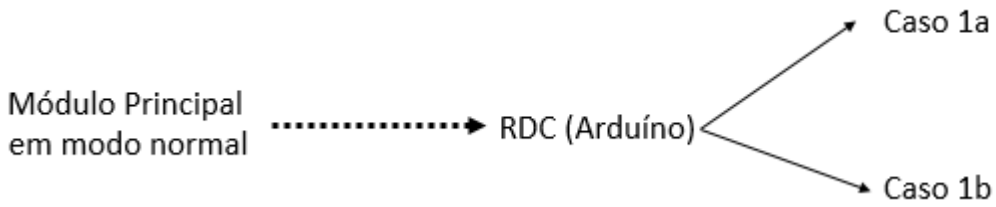
alterações dos dados enviados pela rede Ethernet, que ocorrem durante a comunicação do módulo principal com o RDC é de grande importância.

A comunicação do módulo principal com o RDC ocorre de forma distinta nas diferentes plataformas, como detalhado a seguir.

4.1.1. Comunicação com o RDC implementado no Arduino Uno

A primeira implementação deste trabalho é realizada através da comunicação do módulo principal em modo normal com o RDC implementado no Arduino. Essa implementação está subdividida em dois casos de teste. O primeiro caso, caso 1.a, tem como objetivo o envio do sinal de controle e a análise do efeito da comunicação sobre o valor do mesmo, assim como a consequência desse efeito no sistema. O segundo caso, caso 1.b, tem como objetivo a análise das características temporais da comunicação, Figura 4.5 e Tabela 4.2.

Figura 4.5 - Implementação 1.



Fonte: Produção do Autor.

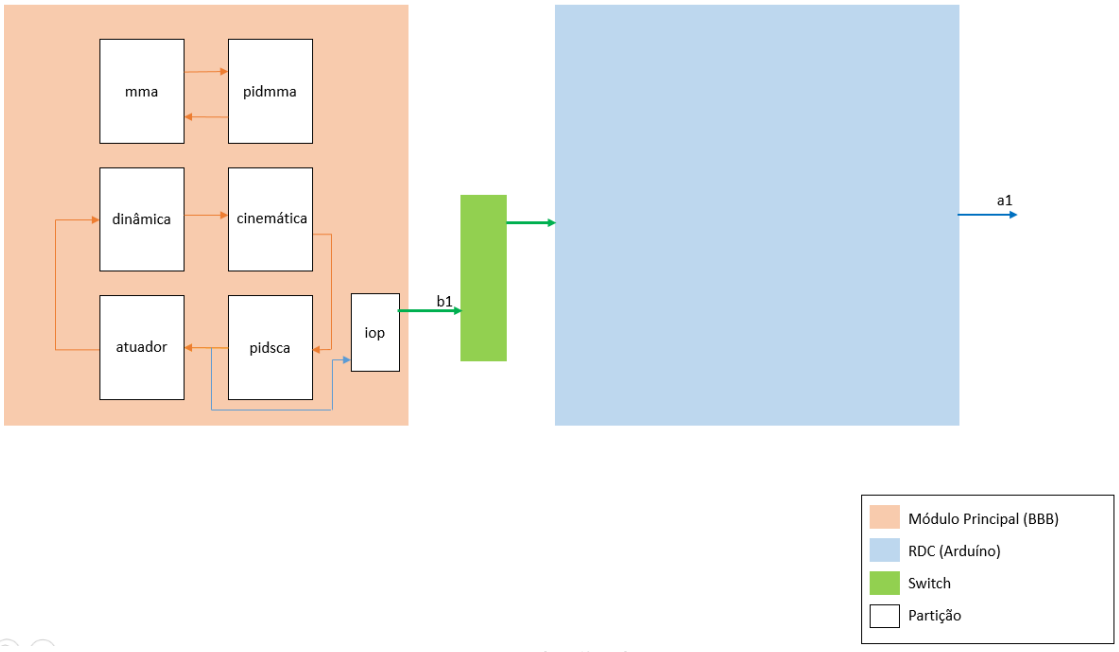
Tabela 4.2 - Implementação 1.

Implementação	Módulo Principal	RDC	Objetivo	Dado enviado
1.a	BBB1	Arduino	Envio do sinal de controle e análise do efeito da comunicação sobre o mesmo.	u (x), u (y), u (z).
1.b	BBB1	Arduino	Análise temporal do envio e do recebimento dos dados.	Número de pulsos do relógio decorridos desde que o sistema é inicializado.

Fonte: Produção do Autor.

A Figura 4.6 abaixo apresenta o esquema de comunicação quando o RDC é implementado na plataforma Arduino:

Figura 4.6 - Comunicação entre o módulo principal e o RDC (Arduino).



Fonte: Produção do Autor.

Os dados oriundos da partição *"pidsca"* e recebidos na partição IOP1, do módulo principal, são gerenciados e enviados para a rede pela partição IOP1. Após o envio dos dados pela rede, eles são recebidos no RDC. Portanto, a comunicação entre o módulo e o RDC, funciona de acordo com a seguinte sequência:

1. PIDSCA → IOP1;
2. IOP1 → Interface de rede;
3. Interface de rede → RDC;

Na primeira etapa, para se conseguir enviar os dados para a partição IOP1 é preciso:

a) Estabelecer um canal entre as duas partições, PIDSCA e IOP1;
b) Configurar e criar uma porta de envio para esse canal na partição emissora PIDSCA;
c) Configurar e criar uma porta de recebimento para esse canal na partição receptora IOP1;
d) Configurar o envio dos dados pela porta da partição emissora PIDSCA;

Na segunda etapa, a comunicação é transparente. A partição emissora PIDSCA envia os dados para a partição receptora IOP1, que automaticamente, envia os dados para a rede. Para isso é preciso:

a) Nomear e identificar o dispositivo lógico, i.e., o RDC;
b) Configurar uma porta remota associada ao dispositivo lógico RDC*;*
c) Configurar o IP e o MAC do dispositivo físico, i.e., o módulo principal BBB1;
d) Configurar uma rota lógica, associada ao dispositivo lógico RDC;
e) Configurar o IP e o MAC do dispositivo lógico RDC;
f) Configurar a porta UDP do BBB1 pela qual os dados são enviados;
g) Configurar o agendamento utilizado no dispositivo físico BBB1.

Na terceira etapa para receber os dados no RDC, é preciso:

a) Configurar o IP e MAC do dispositivo, i.e., RDC;

b) Configurar a porta pela qual se recebe os dados no RDC.

A configuração de porta escolhida no BBB1 e utilizada pelo sistema operacional AIR envia mensagens em fila, de modo que as mensagens enviadas são organizadas em ordem. A operação ocorre em modo FIFO (*First In First Out*) em que a primeira mensagem enviada, colocada na fila, é a primeira ser lida, retirada da fila.

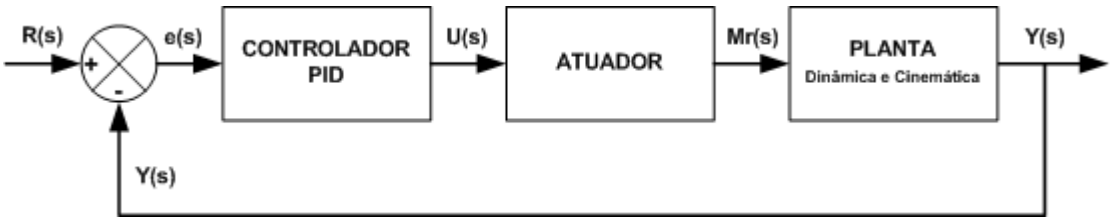
O Arduino deve ser configurado para receber os dados do módulo principal BBB1 corretamente. Estes são do tipo ponto flutuante (*float*) e o Arduino recebe dados como uma sequência de caracteres (*char*) de 1 byte cada; portanto é necessário um processo de conversão de dados.

4.1.1.1. Implementação 1 - Caso de teste A

No primeiro caso de teste desta implementação, realizada através da comunicação do módulo principal BBB1 em modo normal com o RDC implementado no Arduino, o objetivo é o envio do sinal de controle e a análise do efeito da comunicação sobre o valor do mesmo, assim como a consequência desse efeito no sistema.

O módulo principal executa o sistema Massa Mola Amortecedor (MMA) e o modelo compatível com o Sistema de Controle de Atitude (SCA) da PMM. O sistema utilizado para enviar dados ao RDC é o modelo compatível com o Sistema de Controle de Atitude (SCA) da PMM, que tem como função controlar a posição do satélite em relação a um sistema de coordenadas de referência, cuja a origem é o seu centro de massa, (MORAES, 2017). O modelo compatível com o Sistema de Controle de Atitude (SCA) da PMM é representado na Figura 4.7.

Figura 4.7 - Diagrama de blocos do modelo compatível com o SCA.



Fonte: Moraes (2017).

Nessa implementação, os dados enviados para o RDC, são as saídas dos controladores utilizados para cada eixo de rotação, X, Y e Z, cujos comportamentos dinâmicos e os ganhos são definidas em (MORAES, 2017). As equações a diferenças dos controladores PID de cada eixo, do modelo compatível com o Sistema de Controle de Atitude (SCA) da PMM estão definidas abaixo.

$$u_x(n) = u(n-1) + * 690{,}0145e(n) - 1338{,}0359 * e(n-1) + 648{,}7214 * e(n-2) \quad (4.1)$$

$$u_y(n) = u(n-1) + 848{,}1068 * e(n) - 1643{,}0282 * e(n-1) + 795{,}6214 * e(n-2) \quad (4.2)$$

$$u_z(n) = u(n-1) + 743{,}1398 * e(n) - 1440{,}5255 * e(n-1) + 698{,}0857 * e(n-2) \quad (4.3)$$

Tabela 4.3 - Ganhos PID.

		Ganhos		
		K_p	K_d	K_i
Eixos	X	40,5931	454,1050	1,0000
	Y	51,7854	556,9350	1,0000
	Z	44,3541	488,6600	1,0000

Fonte: Moraes (2017).

A cada iteração o sinal de controle para cada eixo é enviado para a rede: X, Y e Z, respectivamente.

De acordo com a modelagem do sistema e a discretização das equações de estado realizadas em (MORAES, 2017), as equações a diferenças que representam o comportamento físico do sistema MMA são:

$$x1(n) = 0{,}9759 * x1(n-1) + 0{,}69437 * x2(n-1) + 0{,}2440 * u\,(n-1) \quad (4.4)$$

$$x2(n) = -0{,}0685 * x1(n-1) + 0{,}9759 * x2(n-1) + 0{,}6944 * u\,(n-1) \quad (4.5)$$

$$y\,(n) = x1(n) \quad (4.6)$$

O comportamento físico e o valor das constantes do controlador PID do sistema MMA são definidas em (MORAES, 2017).

Kp = 0,1512

Ti = 1

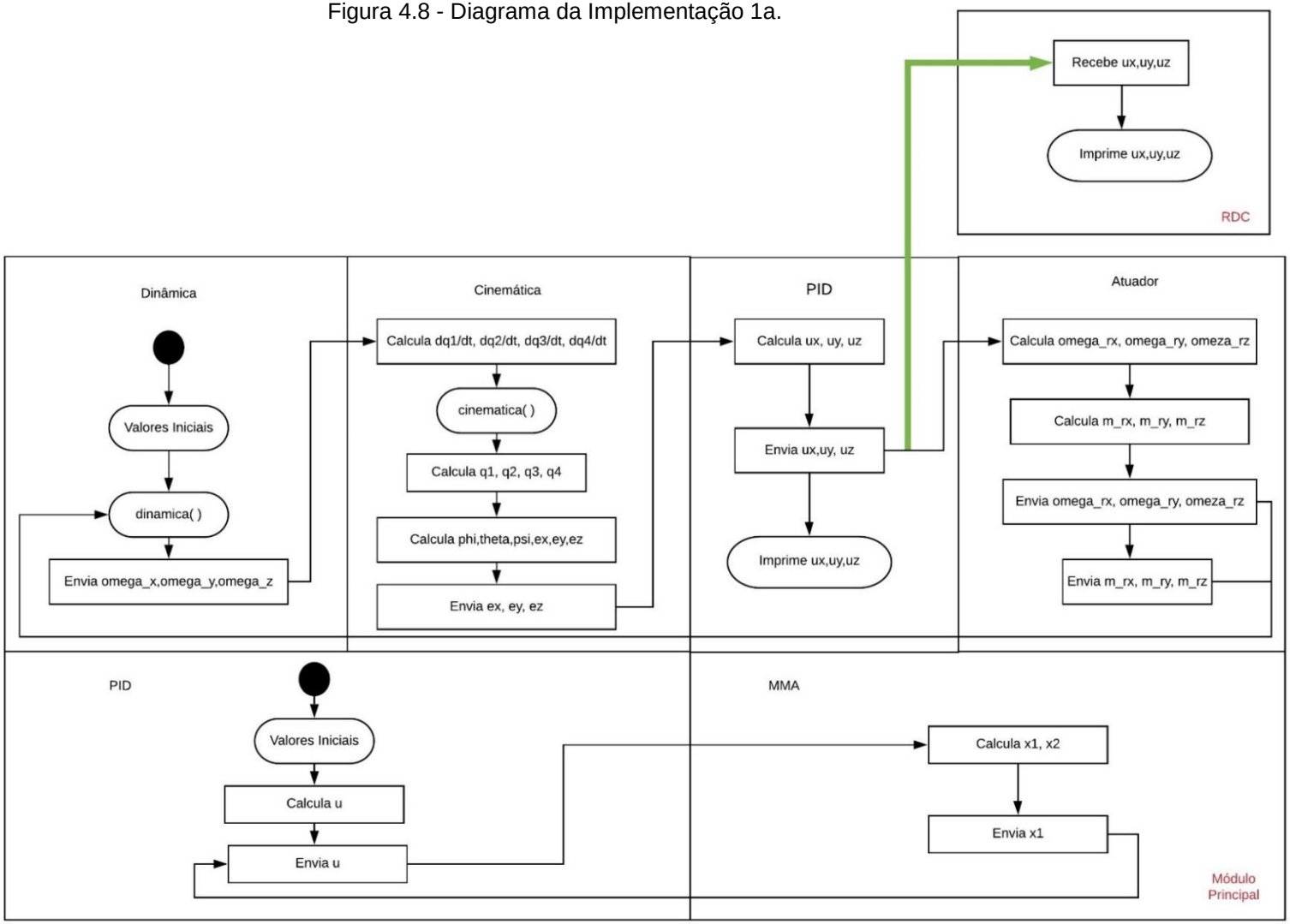
Td = 6,1739

A equação a diferenças do controlador PID do sistema MMA está definida abaixo.

$$u(n) = u(n-1) + 1{,}5906 * e(n) - 2{,}8183 * e(n-1) + 1{,}3336 * e(n-2) \quad (4.7)$$

O diagrama a seguir, Figura 4.8, apresenta a lógica implementada no demonstrador DIMA.

Figura 4.8 - Diagrama da Implementação 1a.



Fonte: Produção do Autor.

4.1.1.2. Implementação 1 - Caso de teste B

No segundo caso de teste desta implementação, realizada através da comunicação do módulo principal BBB1 em modo normal com o RDC implementado no Arduino, o objetivo é a análise das características temporais da comunicação.

O módulo principal executa o sistema Massa Mola Amortecedor (MMA) e o modelo compatível com o sistema de controle (SCA) da PMM. O sistema utilizado para enviar dados ao RDC é o modelo compatível com o sistema de controle (SCA) da PMM.

Para realizar a análise, o dado enviado a cada iteração, pela partição "*pidsca*" do modelo compatível com o sistema de controle (SCA) da PMM do módulo principal BBB1, para o RDC é a saída de uma função do sistema operacional que retorna o número de pulsos (*ticks*) do relógio decorridos desde que o sistema é inicializado.

Os sistemas do módulo principal BBB1 estão configurados da seguinte maneira:

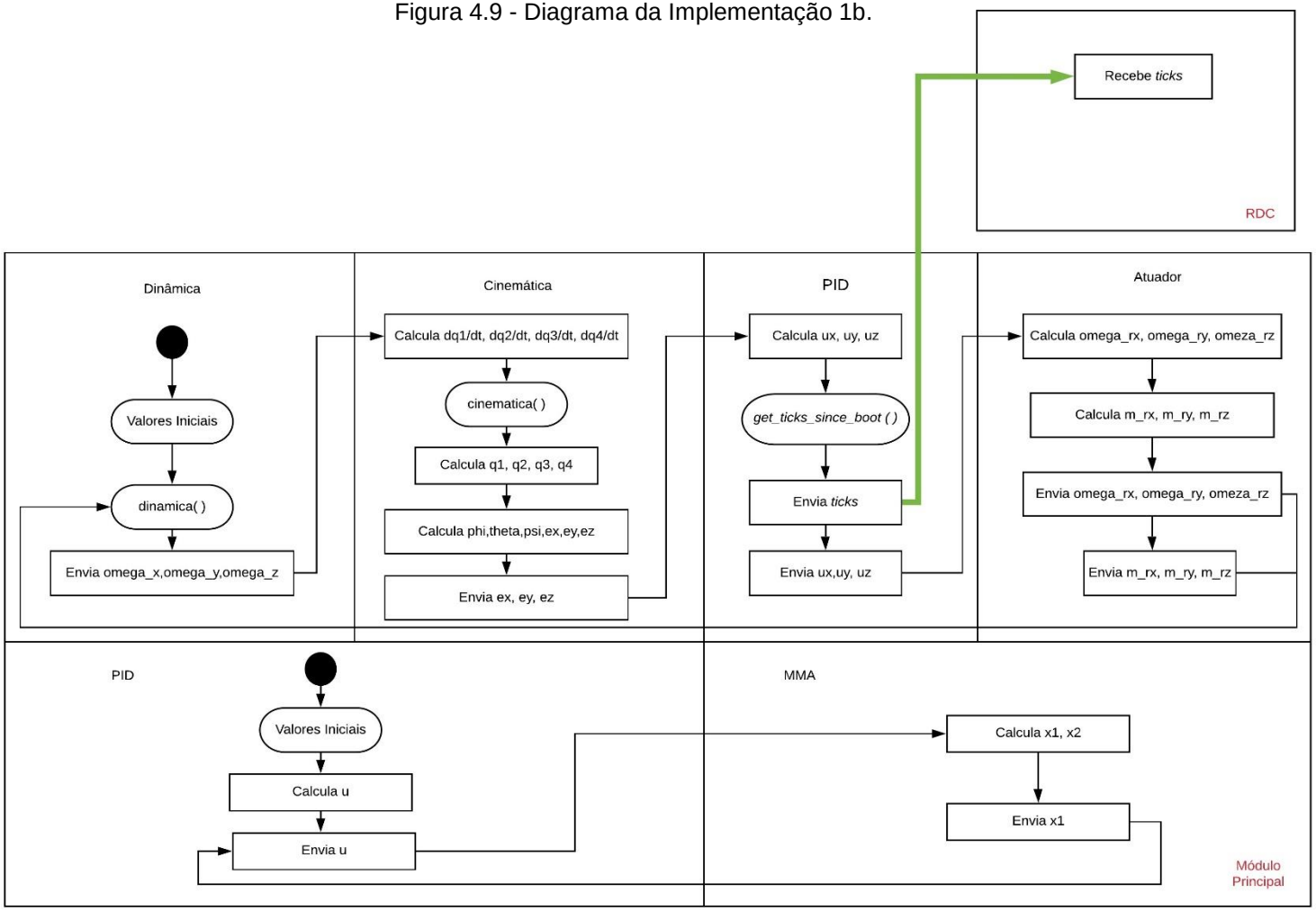
a) Pulsos por segundo = 1000;

b) Cada partição é executada durante 0.1s com período de 0.7s.

No RDC, para saber o momento em que ele recebe cada mensagem, é utilizada uma função própria do Arduino, que retorna o número de milissegundos passados desde que a plataforma começa a executar o programa. Então, no RDC, consegue-se obter a mensagem enviada pelo módulo principal BBB1 e o momento em que essa mensagem é recebida.

O diagrama a seguir, Figura 4.9, apresenta a lógica implementada no demonstrador DIMA.

Figura 4.9 - Diagrama da Implementação 1b.



Fonte: Produção do Autor.

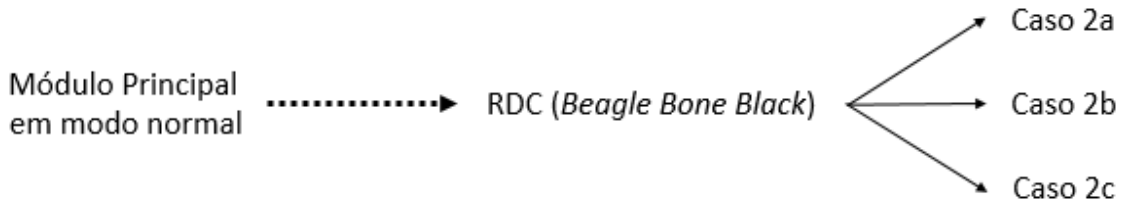
4.1.2. Comunicação com o RDC implementado no *Beagle Bone Black* BBB2

A segunda implementação deste trabalho é realizada através da comunicação do módulo principal BBB1 em modo normal com o RDC implementado na plataforma de desenvolvimento *Beagle Bone Black* BBB2. Essa implementação tem como objetivo a análise das características temporais da comunicação.

O RDC implementado na plataforma de desenvolvimento *Beagle Bone Black* BBB2 permite escolha da frequência de recebimento dos dados pela rede, devido à plataforma permitir embarcar o sistema operacional em tempo real AIR, e também analisar se o comportamento do RDC ao receber os dados é semelhante ao do RDC implementado no Arduino Uno. O sistema operacional AIR possui um agendador que possibilita escolher o período com o que a interface de rede recebe os dados.

Esta implementação está subdividida em três casos de teste. O primeiro caso, caso 2a, tem como objetivo a análise das características temporais da comunicação quando o recebimento dos dados realizado no RDC tem uma frequência maior do que a frequência de envio dos dados. O segundo caso, caso 2b, tem como objetivo a análise das características temporais da comunicação quando o recebimento dos dados realizado no RDC tem uma frequência igual à frequência de envio dos dados. O terceiro caso, caso 2c, tem como objetivo a análise das características temporais da comunicação quando o recebimento dos dados realizado no RDC tem uma frequência menor do que a frequência de envio dos dados, conforme a Figura 4.10 e a Tabela 4.4.

Figura 4.10 - Implementação 2.



Fonte: Produção do Autor.

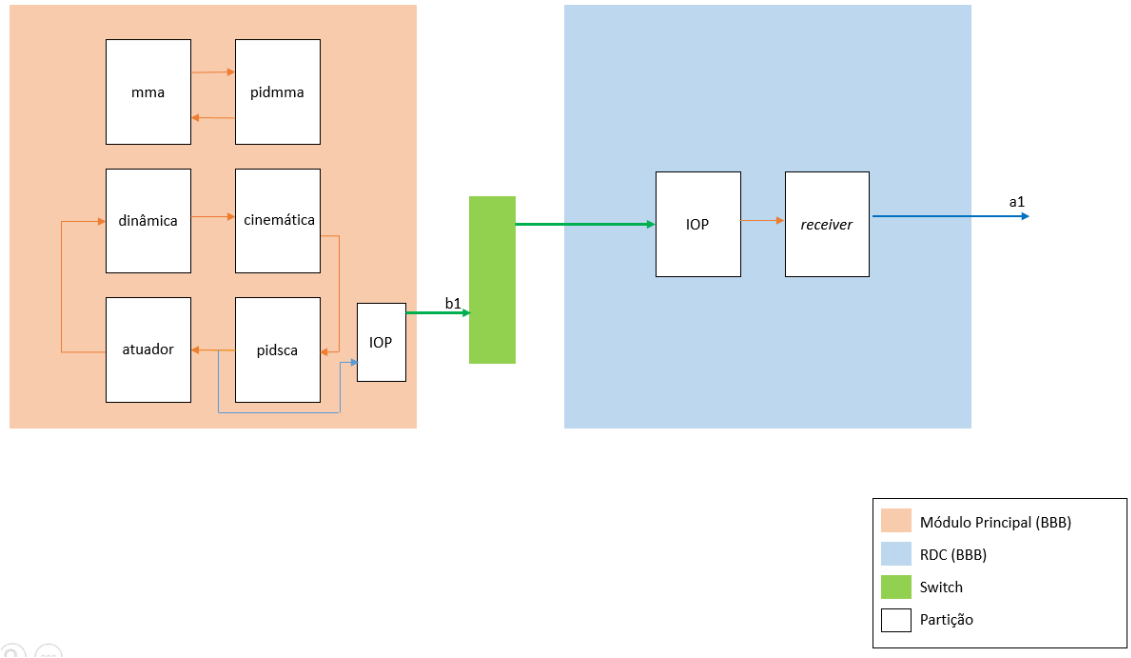
Tabela 4.4 - Implementação 2.

Implementação	Módulo Principal	RDC	Objetivo	Dado enviado
2.a	BBB1	BBB2	Análise temporal do envio e do recebimento dos dados; Frequência de recebimento > Frequência de envio.	Número de pulsos do relógio decorridos desde que o sistema é inicializado.
2.b	BBB1	BBB2	Análise temporal do envio e do recebimento dos dados; Frequência de recebimento = Frequência de envio.	Número de pulsos do relógio decorridos desde que o sistema é inicializado.
2.c	BBB1	BBB2	Análise temporal do envio e do recebimento dos dados. Frequência de recebimento < Frequência de envio.	Número de pulsos do relógio decorridos desde que o sistema é inicializado.

Fonte: Produção do Autor.

A Figura 4.11 apresenta o esquema de comunicação quando o RDC é implementado na plataforma *Beagle Bone Black* BBB2.

Figura 4.11 - Comunicação entre o módulo principal e o RDC (BBB2).



Fonte: Produção do Autor.

Os dados oriundos da partição "*pidsca*" e recebidos na partição IOP1, do módulo principal, são gerenciados e enviados para a rede pela IOP1. Após o envio dos dados pela rede, eles são recebidos na partição IOP2 do RDC e redirecionados para a partição *Receiver*. Portanto, a comunicação entre o módulo principal e o RDC, funciona de acordo com a seguinte sequência:

4. PIDSCA → IOP1;
5. IOP1 → Interface de rede;
6. Interface de rede → IOP2;
7. IOP2 → *Receiver*.

Na primeira etapa para se conseguir enviar os dados para a partição IOP1 é preciso:

a) Estabelecer um canal entre as duas partições, PIDSCA e IOP1;

b) Configurar e criar uma porta de envio para esse canal na partição emissora PIDSCA;

c) Configurar e criar uma porta de recebimento para esse canal na partição receptora IOP1;

d) Configurar o envio dos dados pela porta da partição emissora PIDSCA;

Na segunda etapa, a comunicação é transparente. A partição emissora PIDSCA envia os dados para a partição receptora IOP1, que automaticamente, envia os dados para a rede. Para isso é preciso:

a) Nomear e identificar o dispositivo lógico, i.e., o RDC;

b) Configurar uma porta remota associada ao dispositivo lógico RDC*;*

c) Configurar o IP e o MAC do dispositivo físico, i.e., o módulo principal BBB2;

d) Configurar uma rota lógica, associada ao dispositivo lógico RDC;

e) Configurar o IP e o MAC do dispositivo lógico RDC;

f) Configurar a porta UDP pela qual os dados são enviados pelo BBB1;

g) Configurar o agendamento utilizado no dispositivo físico BBB1.

Na terceira etapa, mais uma vez, do lado do receptor RDC, esta comunicação é transparente. Neste caso, do lado do receptor RDC, o dispositivo físico é o próprio RDC e o dispositivo lógico, é o módulo principal BBB1. Na terceira etapa é preciso:

a) Nomear e identificar o dispositivo lógico, i.e., o módulo principal;

b) Configurar uma porta remota **não** associada ao dispositivo lógico, o módulo principal*;*

c) Configurar o IP e o MAC do dispositivo físico, i.e., o RDC;

d) Configurar uma rota lógica, associada ao dispositivo lógico, o módulo principal;

e) Configurar o IP e o MAC do dispositivo lógico, o módulo principal;

f) Configurar a porta UDP pela qual os dados são recebidos no RDC;

g) Configurar o agendamento utilizado no dispositivo físico BBB2.

Na última etapa, para recebermos os dados, na partição receptora *Receiver*, vindos da IOP2 é preciso:

a) Estabelecer um canal entre as duas partições, IOP2 e *Receiver*;

b) Configurar e criar uma porta de recebimento para esse canal na partição receptora *Receiver*;

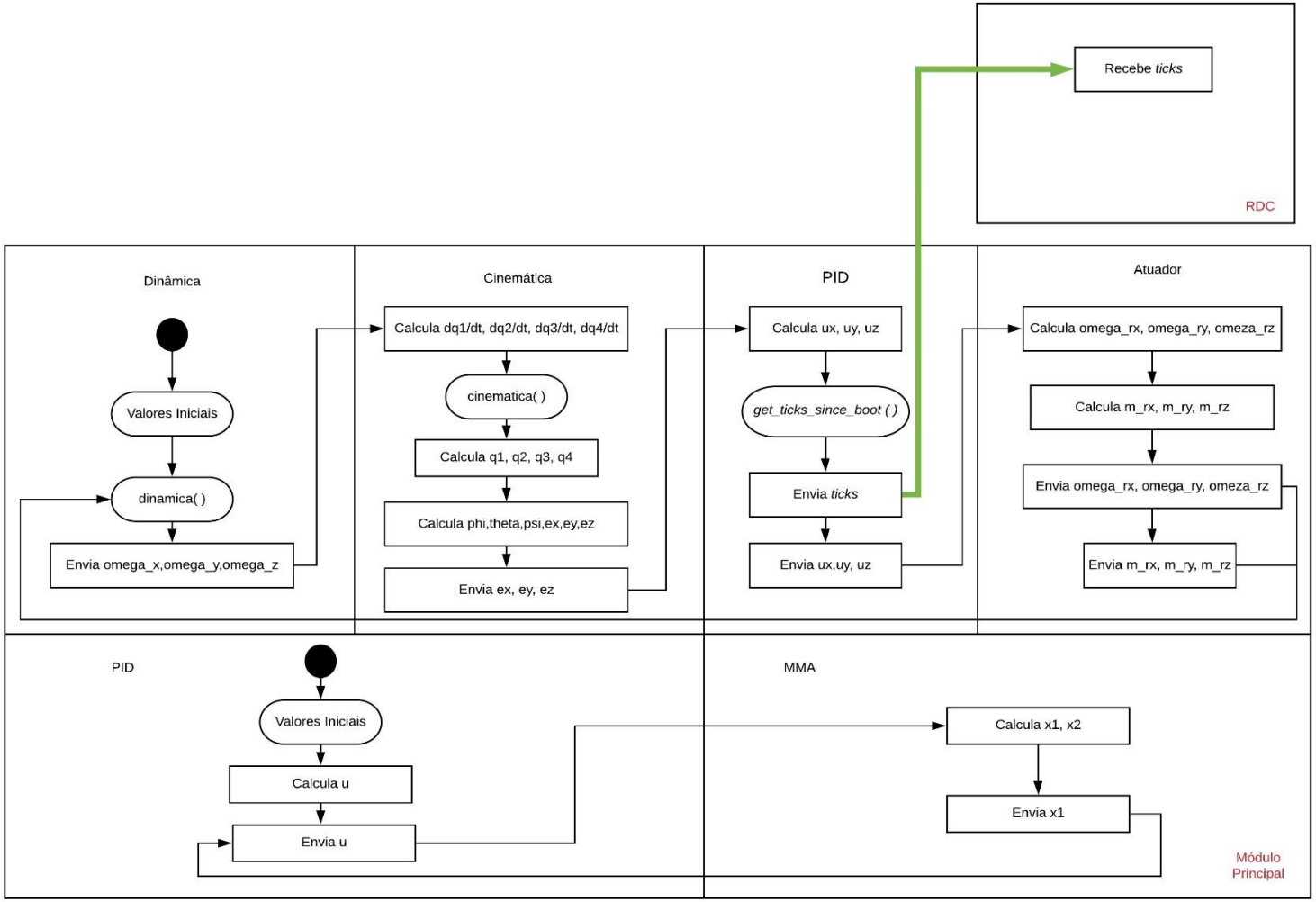
c) Configurar e criar uma porta de envio para esse canal na partição IOP2;

d) Configurar o recebimento dos dados pela porta da partição receptora *Receiver*;

A configuração de porta escolhida e utilizada no sistema operacional AIR envia mensagens em fila, de modo que as mensagens enviadas são organizadas em ordem. A operação ocorre em modo FIFO (*First In First Out*) em que a primeira mensagem enviada, colocada na fila, é a primeira ser lida, retirada da fila.

O diagrama com a lógica implementada nos três casos dessa implementação, Figura 4.13, no demonstrador DIMA é o mesmo apresentado, do ponto de vista lógico, na implementação 1 – caso de teste B, Figura 4.9.

Figura 4.12 - Diagrama das Implementações 2a,2b e 2c.



Fonte: Produção do Autor.

4.1.2.1. Implementação 2 - Caso de teste A

No primeiro caso de teste desta implementação, realizada através da comunicação do módulo principal em modo normal com o RDC implementado no *Beagle Bone Black* BBB2, o objetivo é a análise das características temporais da comunicação quando o recebimento dos dados realizado no RDC tem uma frequência maior do que a frequência de envio dos dados.

O módulo principal executa o sistema Massa Mola Amortecedor (MMA) e o modelo compatível com o Sistema de Controle de Atitude (SCA) da PMM. O sistema utilizado para enviar dados ao RDC é o modelo compatível com o Sistema de Controle de Atitude (SCA) da PMM.

Para realizar a análise, o dado enviado a cada iteração, pela partição "*pidsca*" do modelo compatível com o Sistema de Controle de Atitude (SCA) da PMM do módulo principal, para o RDC é a saída de uma função do sistema operacional que retorna o número de pulsos (*ticks*) do relógio decorridos desde que o sistema é inicializado.

Os sistemas do módulo principal estão configurados da seguinte maneira:

a) Pulsos por segundo = 1000;

b) Cada partição é executada durante 0.1s com período de 0.7s.

No RDC, para saber o momento em que ele recebe cada mensagem, é utilizada a saída de uma função do sistema operacional que retorna o número de *ticks* do relógio decorridos desde que o sistema é inicializado.

O RDC está configurado da seguinte maneira:

a) Pulsos por segundo = 1000;

b) Cada partição é executada durante 0.1s com período de 0.2s;

Então, no RDC, consegue-se obter a mensagem enviada pelo módulo principal e o momento em que essa mensagem é recebida.

4.1.2.2. Implementação 2 - Caso de teste B

No segundo caso de teste desta implementação, realizada através da comunicação do módulo principal em modo normal com o RDC implementado no *Beagle Bone Black* BBB2, o objetivo é a análise das características temporais da comunicação quando a frequência de recebimento dos dados realizado no RDC é a mesma da frequência de envio dos dados.

O módulo principal executa o sistema Massa Mola Amortecedor (MMA) e o modelo compatível com o Sistema de Controle de Atitude (SCA) da PMM. O sistema utilizado para enviar dados ao RDC é o modelo compatível com o Sistema de Controle de Atitude (SCA) da PMM.

Para realizar a análise, o dado enviado a cada iteração, pela partição "*pidsca*" do modelo compatível com o Sistema de Controle de Atitude (SCA) da PMM do módulo principal, para o RDC é a saída de uma função do sistema operacional que retorna o número de pulsos (*ticks*) do relógio decorridos desde que o sistema é inicializado.

Os sistemas do módulo principal estão configurados da seguinte maneira:

a) Pulsos por segundo = 1000;

b) Cada partição é executada durante 0.1s com período de 0.7s.

No RDC, para saber o momento em que ele recebe cada mensagem, é utilizada a saída de uma função do sistema operacional que retorna o número de *ticks* do relógio decorridos desde que o sistema é inicializado.

O RDC está configurado da seguinte maneira:

a) Pulsos por segundo = 1000;

b) Cada partição é executada durante 0.1s com período de 0.7s;

Então, no RDC, consegue-se obter a mensagem enviada pelo módulo principal e o momento em que essa mensagem é recebida.

4.1.2.3. Implementação 2 - Caso de teste C

No terceiro caso de teste desta implementação, o objetivo é a análise das características temporais da comunicação quando a frequência de recebimento dos dados realizado no RDC é menor que a frequência de envio dos dados.

O módulo principal executa o sistema Massa Mola Amortecedor (MMA) e o modelo compatível com o Sistema de Controle de Atitude (SCA) da PMM. O sistema utilizado para enviar dados ao RDC é o modelo compatível com o Sistema de Controle de Atitude (SCA) da PMM.

Para realizar a análise, o dado enviado a cada iteração, pela partição "*pidsca*" do modelo compatível com o Sistema de Controle de Atitude (SCA) da PMM do módulo principal, para o RDC é a saída de uma função do sistema operacional que retorna o número de pulsos (*ticks*) do relógio decorridos desde que o sistema é inicializado.

Os sistemas do módulo principal estão configurados da seguinte maneira:

a) Pulsos por segundo = 1000;

b) Cada partição é executada durante 0.1s com período de 0.7s.

No RDC, para saber o momento em que ele recebe cada mensagem, é utilizada a saída de uma função do sistema operacional que retorna o número de *ticks* do relógio decorridos desde que o sistema é inicializado.

O RDC está configurado da seguinte maneira:

a) Pulsos por segundo = 1000;

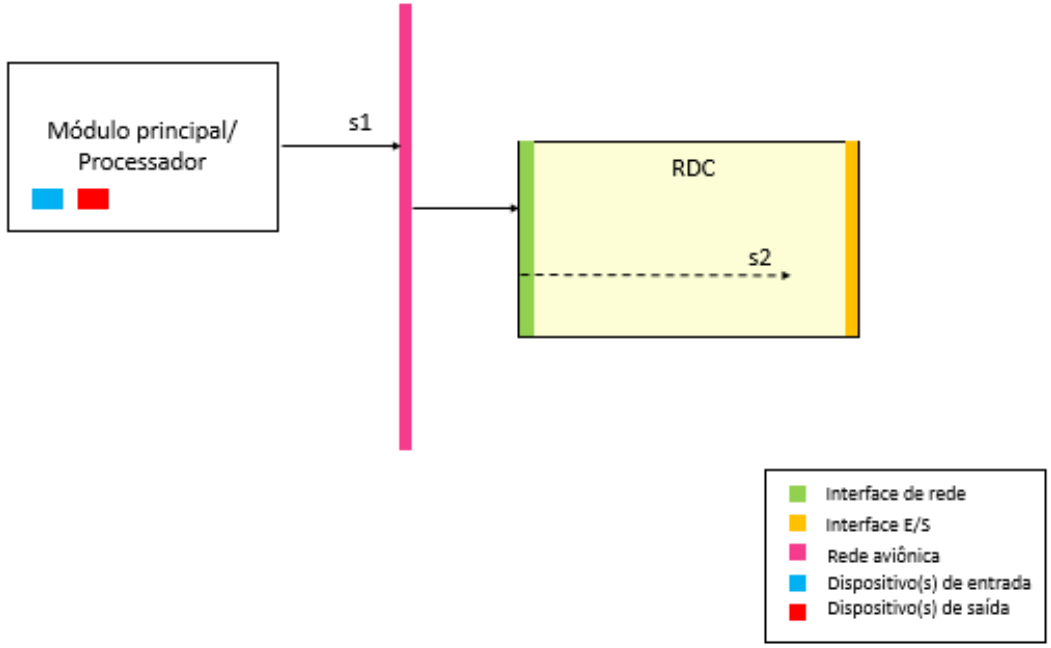
b) Cada partição é executada durante 0.1s com período de 1.0s;

Então, no RDC, consegue-se obter a mensagem enviada pelo módulo principal e o momento em que essa mensagem é recebida.

4.2. Módulo principal no modo com falha

No modo com falha de operação, o módulo principal do demonstrador DIMA se comunica com o RDC conforme apresentado na Figura 4.13, que demonstra o fluxo de sinal da rede, em que *s1* é o sinal oriundo do módulo principal, e *s2* é o sinal recebido pelo RDC.

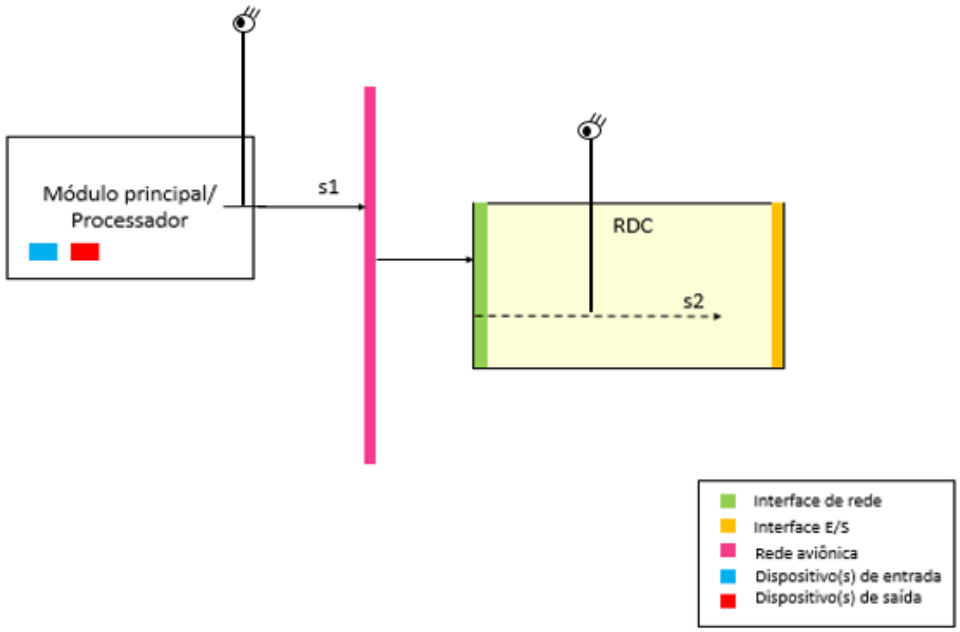
Figura 4.13 - Módulo principal e RDC.



Fonte: Produção do Autor.

A análise das possíveis alterações dos dados enviados pela rede Ethernet, é feita por meio da comparação entre o sinal enviado, *s1*, e o sinal recebido, *s2*. O sinal enviado, *s1*, é observado através da interface de comunicação serial RS232 do *Beagle Bone Black* BBB1, gerenciada pelo software do terminal serial PuTTY. A observação do sinal recebido, *s2*, é realizada através do monitor serial do Arduino. As observações dos sinais estão representadas pelo “ícone do olho” na Figura 4.14 a seguir.

Figura 4.14 - Observação do sinal entre o módulo principal e o RDC.



Fonte: Produção do Autor.

O módulo principal, no trabalho de (MORAES, 2017) é composto pelo sistema MMA, composto por duas partições: “mma” e “pidmma”; e pelo modelo compatível com sistema SCA da PMM, composto por seis partições: mma, pidmma, dinâmica, cinemática, atuador e pidsca; essas partições são executadas no sistema operacional AIR durante 0.1s com período de 0.6s. No modo com falha é inserido um erro de computação no sistema SCA para que se comprove que as demais partições continuam funcionando e que o sistema é capaz de se reconfigurar tratando a condição de falha inserida.

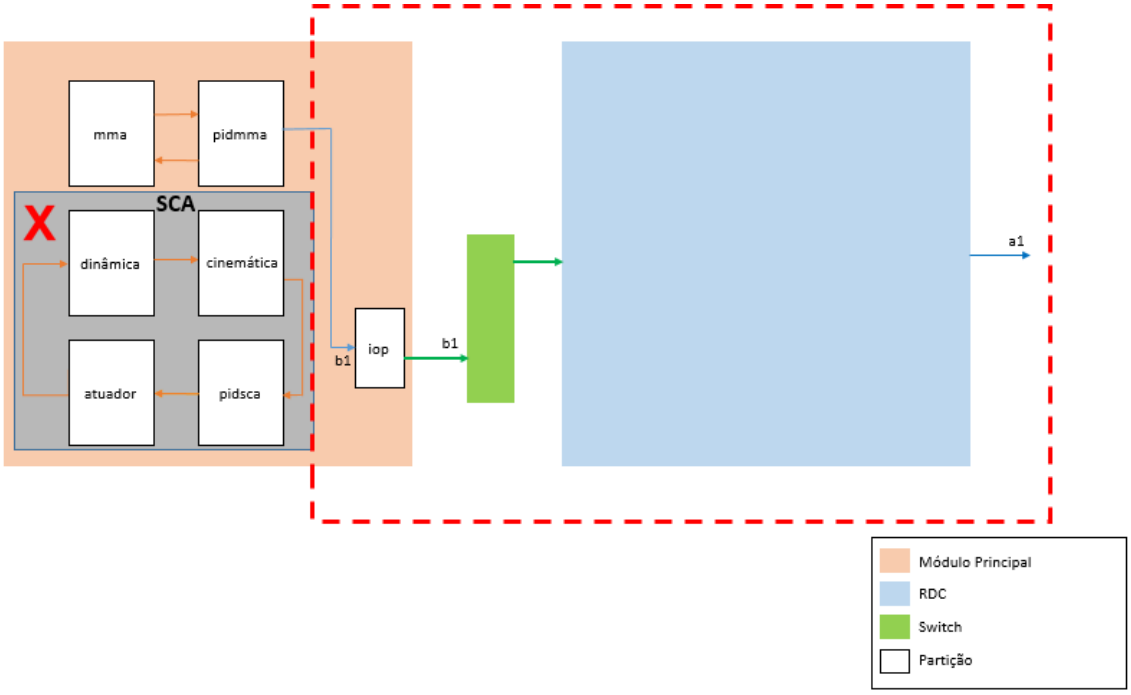
Para que haja a comunicação entre o RDC e o módulo principal, via rede Ethernet, é necessário criar uma nova partição (IOP1) no módulo principal, Figura 4.15. A partição IOP1 apenas gerencia essa comunicação. Observa-se que, o sinal *b1* recebido por essa partição é o mesmo sinal que é enviado à rede Ethernet.

A nova partição, IOP1, é executada no sistema operacional AIR durante 0.1s. Ao adicioná-la, o período de todas as partições passa a ser de 0.7s. O passo de integração dos sistemas SCA e MMA definido por (MORAES, 2017) é de 0.6s.

Para execução de todas as partições, incluindo a IOP1, o passo de integração é estimado em 0.7s.

Na Figura 4.15 é possível observar, na área delimitada pelo quadrado vermelho tracejado, os acréscimos ao sistema DIMA do trabalho do André Moraes, (MORAES, 2017): o RDC e a comunicação via rede Ethernet.

Figura 4.15 - Comunicação entre o módulo principal e o RDC.



Fonte: Produção do Autor.

A partição "*pidmma*" do sistema MMA é a responsável pelo envio dos dados para o RDC. O envio é realizado a fim de comprovar que o erro de computação inserido no sistema SCA e sua reconfiguração ao tratar a condição de falha inserida, não interferem no funcionamento e na comunicação do sistema MMA com o RDC.

O erro de computação se trata de um acesso a endereço inválido de memória, requerido através do endereçamento de um ponteiro nulo. Ele é inserido no

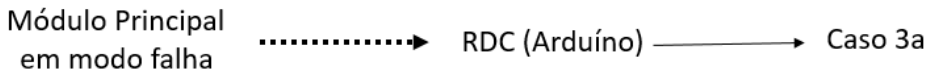
código de uma das partições do sistema SCA, e a função "*Health Monitoring*" do AIR é usada para reconfigurar a partição e isolar a falha, (MORAES, 2017).

A comunicação do módulo principal com o RDC implementado no Arduino ocorre da mesma forma apresentada anteriormente com o módulo principal em modo normal.

4.2.1.1. Implementação 3

A terceira implementação deste trabalho é realizada através da comunicação do sistema MMA do módulo principal no modo com falha com o RDC implementado no Arduino. Essa implementação, Figura 4.16, tem como objetivo comprovar que o erro de computação (apresentado no Capítulo 3) inserido no sistema SCA e a reconfiguração do mesmo, ao tratar a condição de falha inserida, não interferem no funcionamento e na comunicação do sistema MMA com o RDC.

Figura 4.16 - Implementação 3.



Fonte: Produção do Autor.

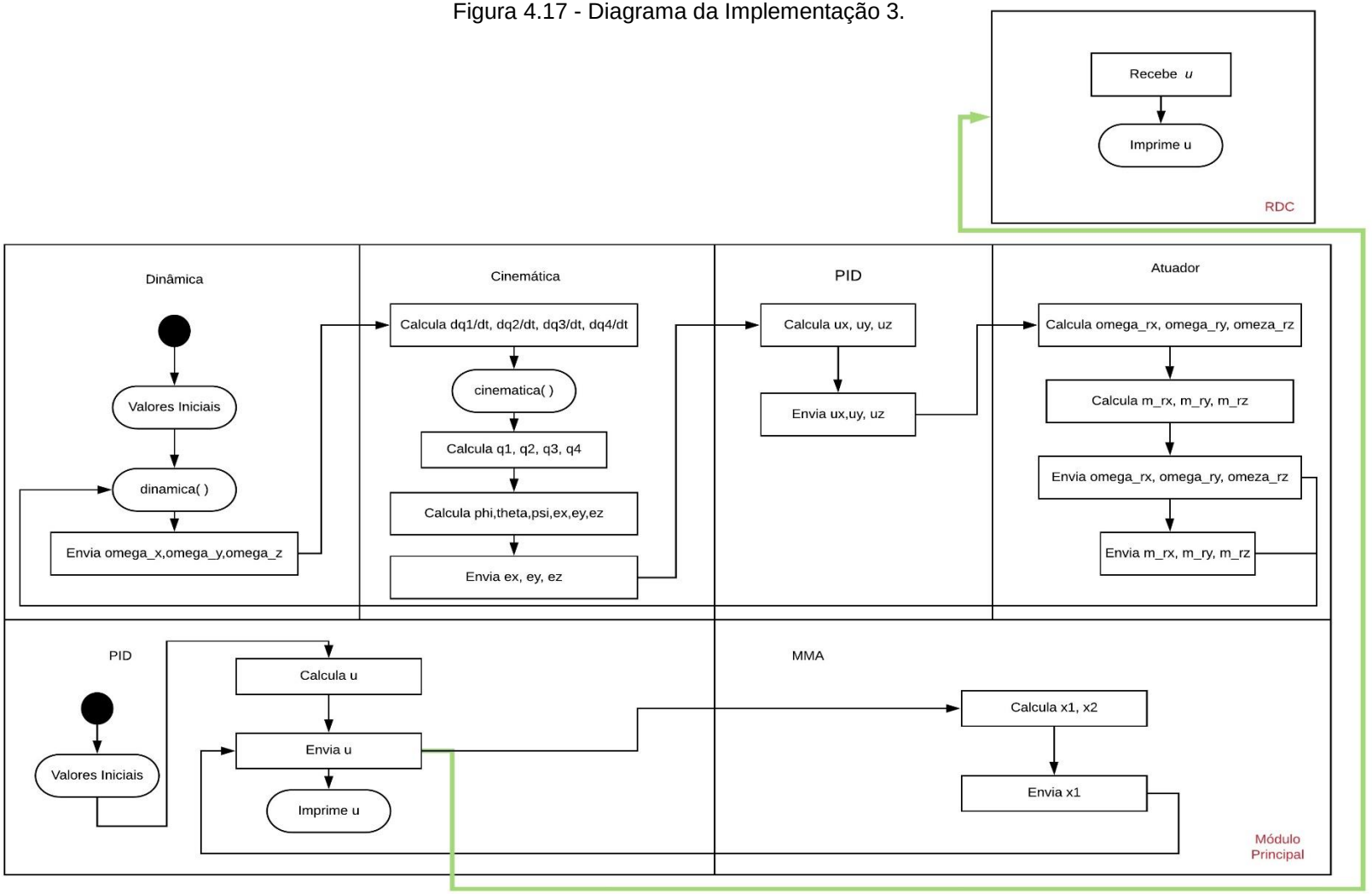
Tabela 4.5 - Implementação 3.

Implementação	Módulo Principal	RDC	Objetivo	Dado enviado
3.a	BBB1	Arduino	Comprovar o isolamento das partições diante de falhas numa delas.	Sinal de controle do Sistema MMA.

Fonte: Produção do Autor.

O diagrama a seguir, Figura 4.17, apresenta a lógica implementada no demonstrador DIMA.

Figura 4.17 - Diagrama da Implementação 3.

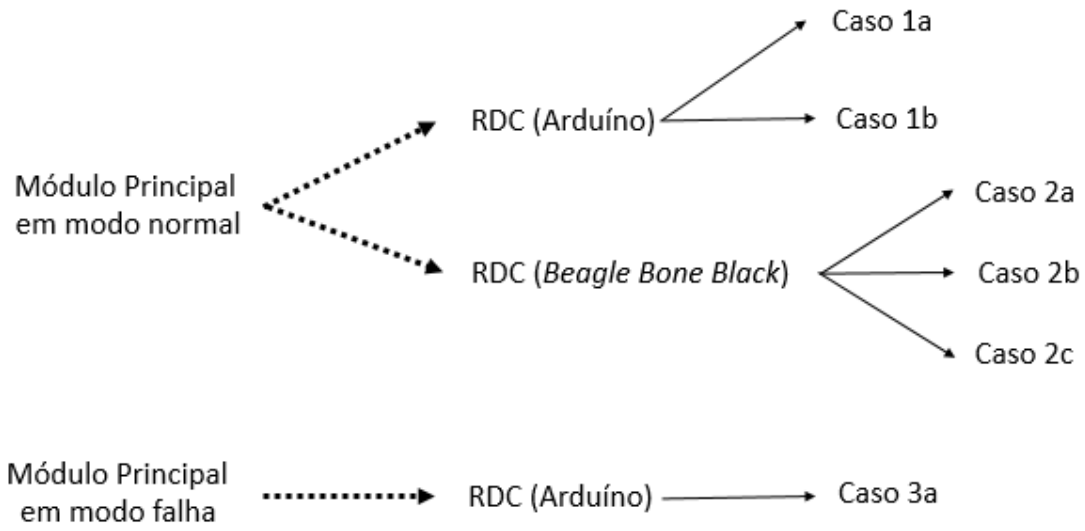


Fonte: Produção do Autor.

5 RESULTADOS E ANÁLISES

Neste Capítulo são apresentadas as análises dos resultados obtidos nos casos de teste propostos nas implementações realizadas neste trabalho, listadas na Figura 5.1. Em cada caso de teste das implementações é realizada uma verificação lógica dos dados enviados pela rede e, quando aplicável, uma análise temporal dos dados enviados e dos dados recebidos.

Figura 5.1 - Árvore de possibilidades de implementação.



Fonte: Produção do Autor.

Na Tabela 5.1, é possível observar os casos de teste das implementações, a plataforma de desenvolvimento utilizada para o módulo principal e para o RDC em cada implementação, assim como o objetivo, e o dado enviado a cada iteração.

Tabela 5.1 - Implementações.

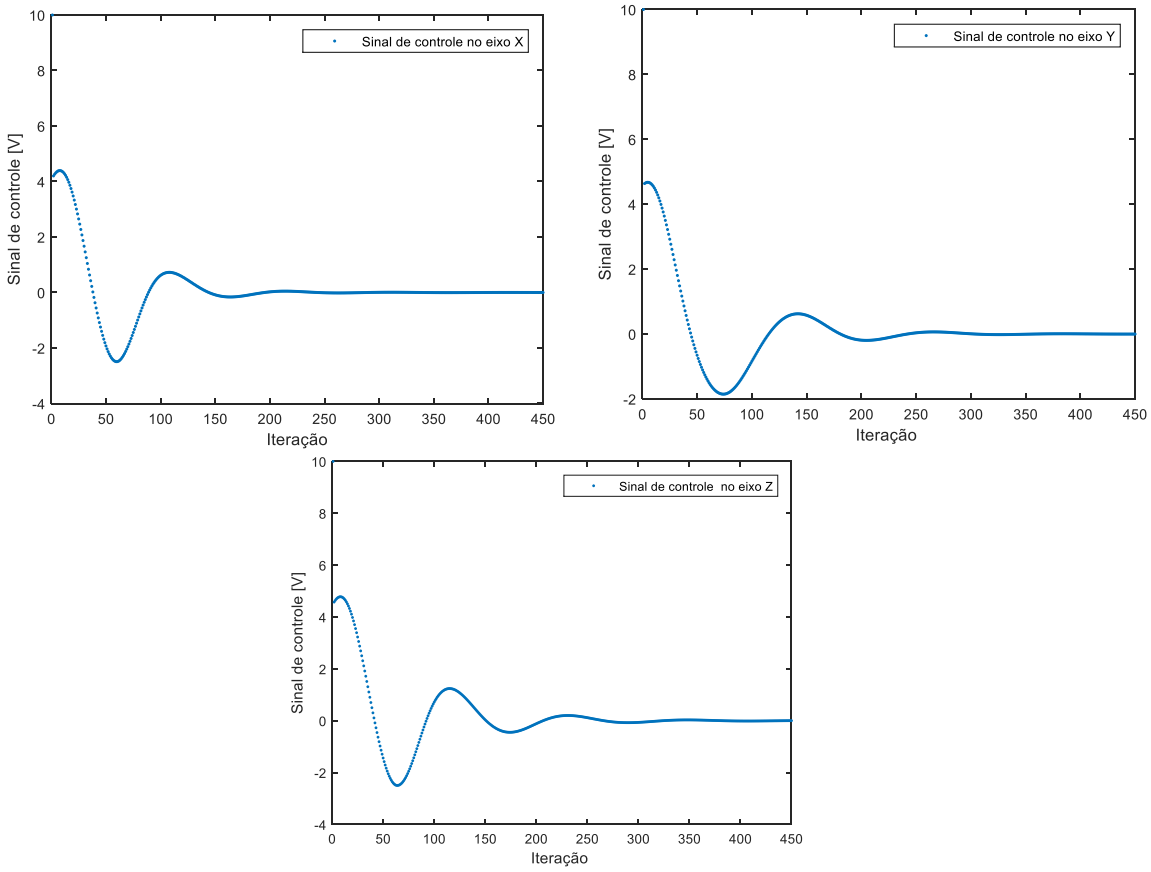
Implementação	Módulo Principal	RDC	Objetivo	Dado enviado
1.a	BBB1	Arduino	Envio do sinal de controle e análise do efeito da comunicação sobre o mesmo.	u (x), u (y), u (z).
1.b	BBB1	Arduino	Análise temporal do envio e do recebimento dos dados.	Número de pulsos do relógio decorridos desde que o sistema é inicializado.
2.a	BBB1	BBB2	Análise temporal do envio e do recebimento dos dados; Frequência de recebimento > Frequência de envio.	Número de pulsos do relógio decorridos desde que o sistema é inicializado.
2.b	BBB1	BBB2	Análise temporal do envio e do recebimento dos dados; Frequência de recebimento = Frequência de envio.	Número de pulsos do relógio decorridos desde que o sistema é inicializado.
2.c	BBB1	BBB2	Análise temporal do envio e do recebimento dos dados. Frequência de recebimento < Frequência de envio.	Número de pulsos do relógio decorridos desde que o sistema é inicializado.
3.a	BBB1 (Modo falha)	Arduino	Comprovar o isolamento das partições diante de falhas numa delas.	Sinal de controle do Sistema MMA.

Fonte: Produção do Autor.

5.1. Implementação 1 - Caso de teste A

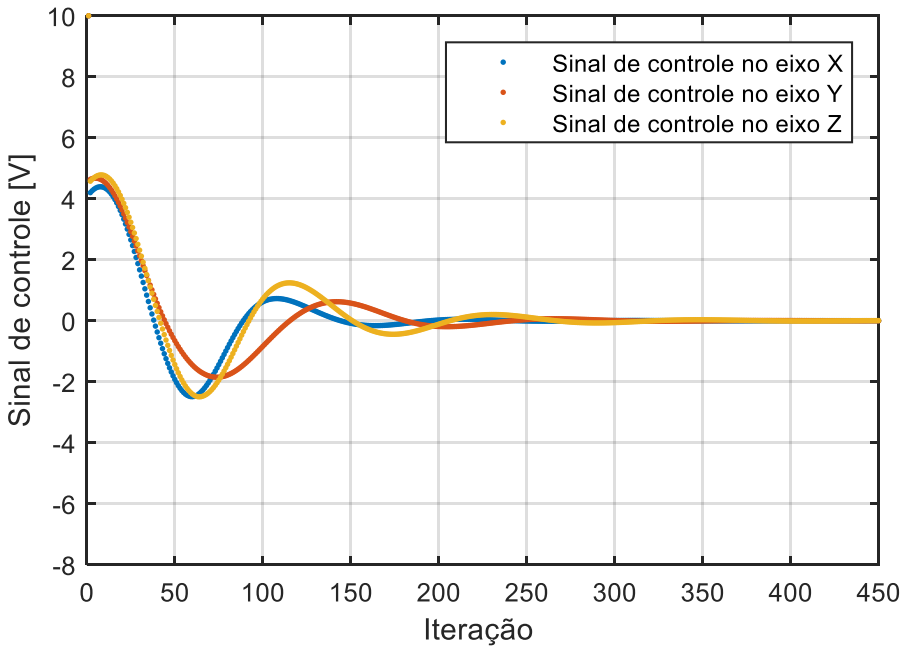
Nesta implementação o módulo principal envia os dados dos sinais de controle dos eixos: X, Y e Z, para o RDC. A cada iteração da partição "*pidsca*" os dados calculados, são enviados em sequência: sinais de controle nos eixos X, Y e Z, respectivamente, ou seja, são enviados 3 dados a cada iteração. Essa implementação tem um total de 1350 dados enviados, 450 dados para cada sinal de controle, onde as mensagens enviadas são organizadas em fila e lidas em ordem (FIFO). Observa-se na Figura 5.2 e na Figura 5.3, os sinais de controle enviados para os eixos X, Y e Z.

Figura 5.2 - Sinais de controle enviados.



Fonte: Produção do Autor.

Figura 5.3 - Sinais de controle enviados.



Fonte: Produção do Autor.

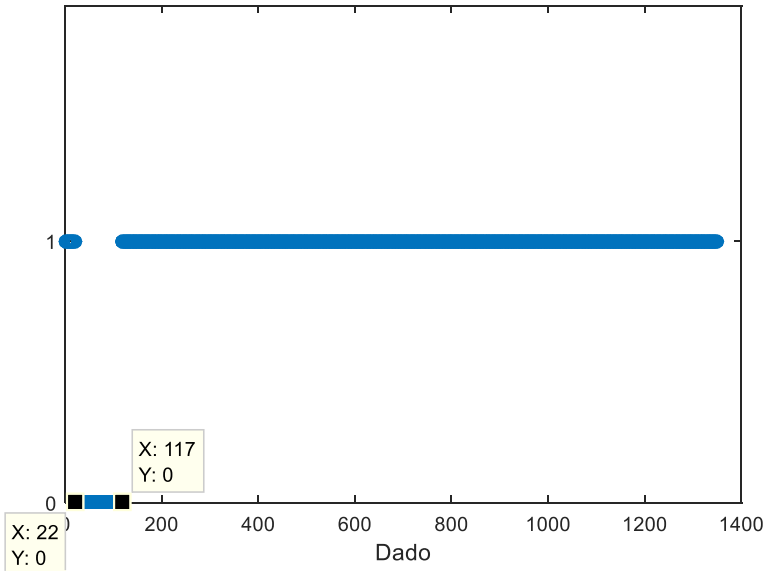
5.1.1. Verificação lógica dos dados

A verificação lógica dos dados é realizada por meio da comparação entre os dados enviados pelo módulo principal e os recebidos pelo RDC. É verificado se os dados são entregues.

Ao se fazer a verificação dos dados recebidos, é possível perceber na Figura 5.4, que alguns dados são perdidos durante a comunicação.

Nesta implementação, os dados identificados como: 1, 2 e 3, fazem parte da primeira iteração, os dados: 4, 5 e 6, fazem parte da segunda iteração, e assim sucessivamente. Logo, no gráifco da Figura 5.4, no eixo X, o dado 22 representa o primeiro dado enviado da oitava iteração, e o dado 117 representa o último dado enviado da trigésima nona iteração. No gráfico da Figura 5.4, no eixo Y, o número 1 representa dados recebidos, e o número 0 representa dados perdidos.

Figura 5.4 - Análise dos dados recebidos.



Fonte: Produção do Autor.

Nota-se que, os dados da oitava iteração até a trigésima nona não são recebidos no RDC.

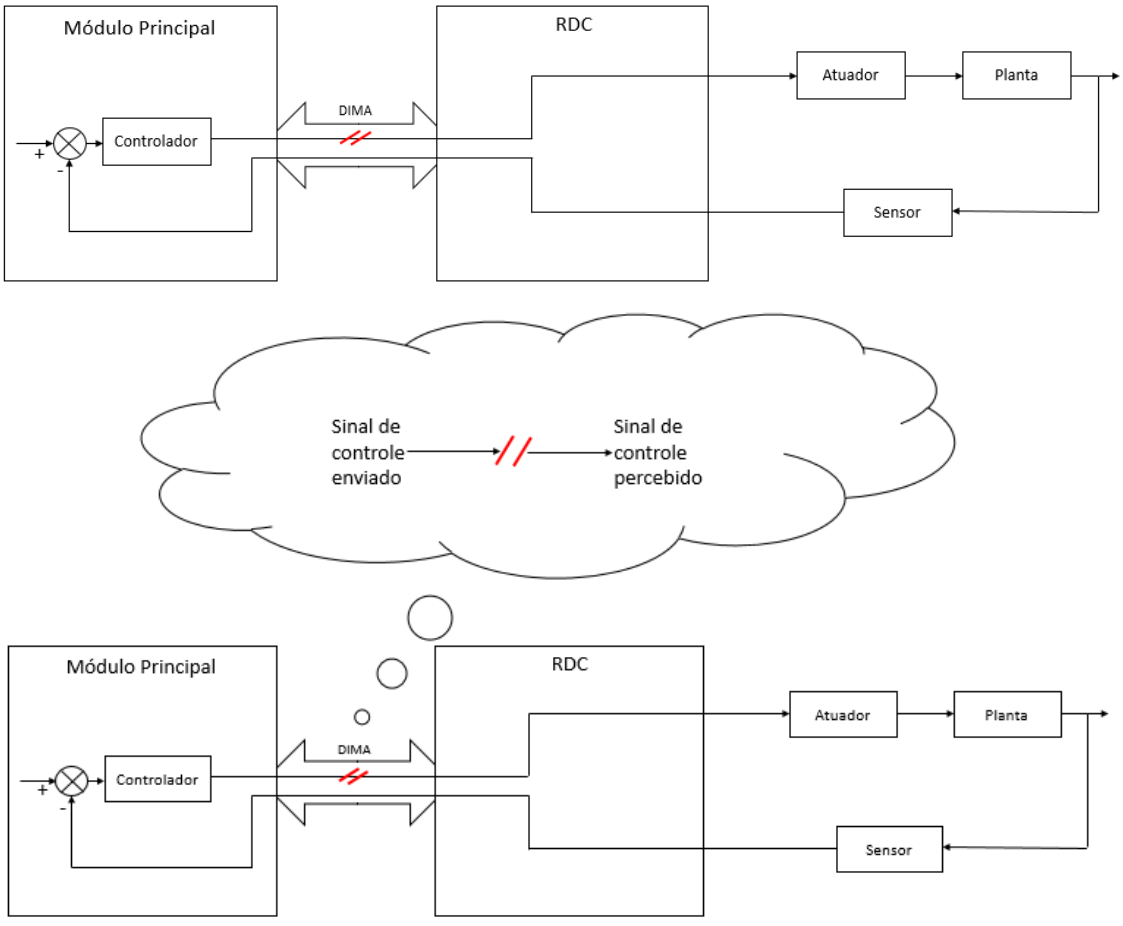
O RDC recebeu 92.8889%,(1254 dados recebidos dos 1350 enviados). É provável que isso aconteça, devido ao tempo inicial, desconhecido e variável que o RDC nota a existência de dados no canal/porta de comunicação e efetua o recebimento dos mesmos.

O fenômeno que faz o RDC não conseguir notar a existência dos dados no canal/porta e efetuar o recebimento dos dados em tempo hábil, ou seja, retirar as mensagens, os dados do canal/porta, faz com que o canal/porta fique cheio e não tenha espaço disponível para receber novas mensagens. Pois, o módulo principal não suspende o envio das mensagens. Como não há espaço no canal/porta, as mensagens enviadas pela partição “pidsca” para IOP1 são perdidas quando a mesma tenta enviar para a rede. Quando há espaço novamente, as mensagens enviadas se referem a um outro valor de sinal de controle. O reenvio dos dados que são perdidos não é solicitado pela rede, pois o protocolo de comunicação utilizado é o UDP.

5.1.2. Análises

A fim de se analisar o efeito da perda dos sinais de controle sobre o sistema, propôs-se simular uma falha na comunicação do sistema SCA com planta executada no módulo principal, BBB1. Essa falha representa a perda de dados entre o módulo principal e o RDC. A falha é introduzida de forma a impedir a atualização dos sinais de controle recebido pela planta. O diagrama de blocos na Figura 5.5 demonstra o ponto de falha em uma arquitetura ideal do demonstrador DIMA.

Figura 5.5 - Ponto de falha.



Fonte: Produção do Autor.

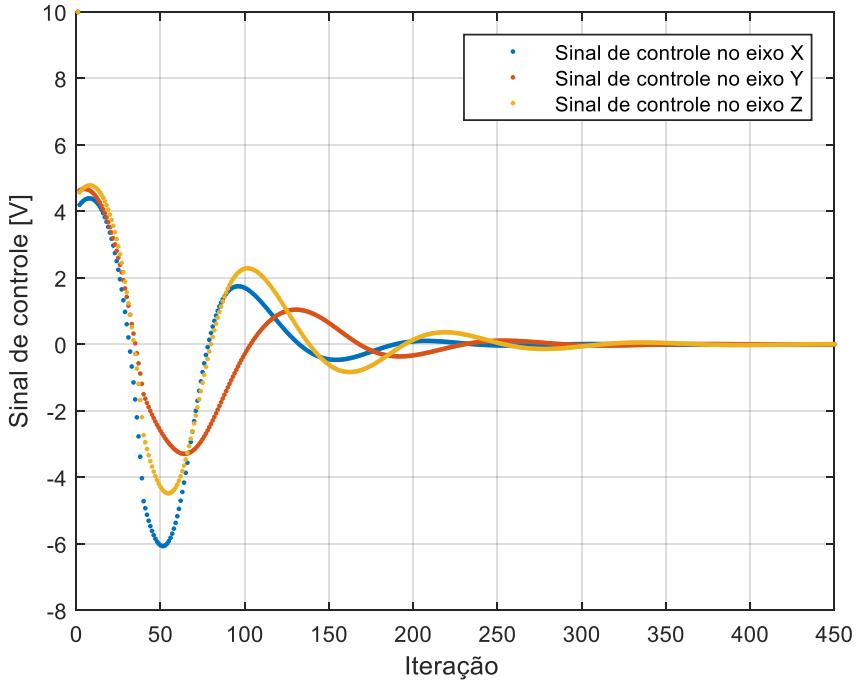
São realizadas duas análises do efeito da perda dos sinais de controle sobre o sistema. As análises diferem nos valores dos sinais de controle percebidos pela planta no momento que ocorre a falha de comunicação. Em seguida, é realizada uma comparação das duas análises.

5.1.2.1. Análise *dead reckoning*

Na análise *dead reckoning*, enquanto os sinais de controle são impedidos de serem atualizados, i.e., no momento que ocorre a falha de comunicação, a planta recebe e mantém o último valor de sinal imediatamente anterior à ocorrência da falha, como na Figura 5.7.

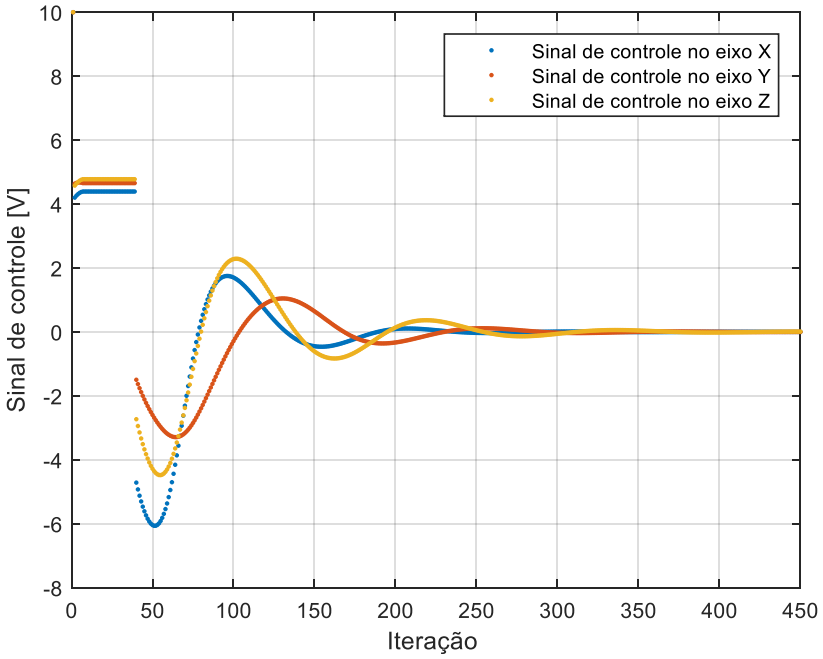
No gráfico da Figura 5.6, é possível observar um aumento na oscilação dos sinais de controle, o que significa perda de desempenho, ao se comparar com os sinais de controle quando não há falha, na Figura 5.3. Isto ocorre, pois, o sistema fica em malha aberta durante a falha (desde o dado 22 até o dado 117- desde a oitava iteração até a trigésima nona iteração), logo o erro, falta de atualização do controle, introduzido se propaga pelo mesmo.

Figura 5.6 - Sinais de controle enviados.



Fonte: Produção do Autor.

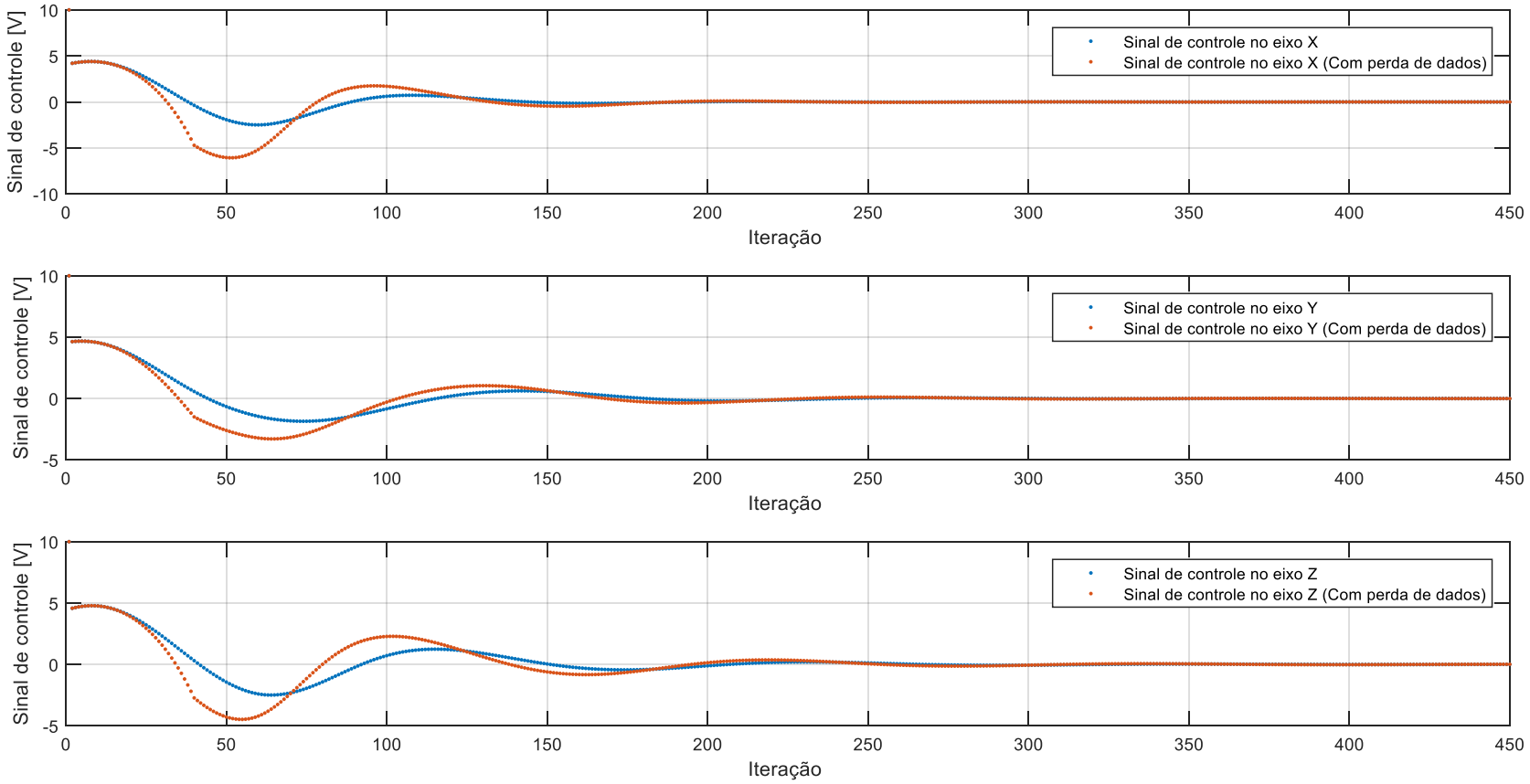
Figura 5.7 - Sinais de controle percebidos.



Fonte: Produção do Autor.

A Figura 5.8 permite comparar o sinal de controle em cada eixo, sem e com a falha, respectivamente.

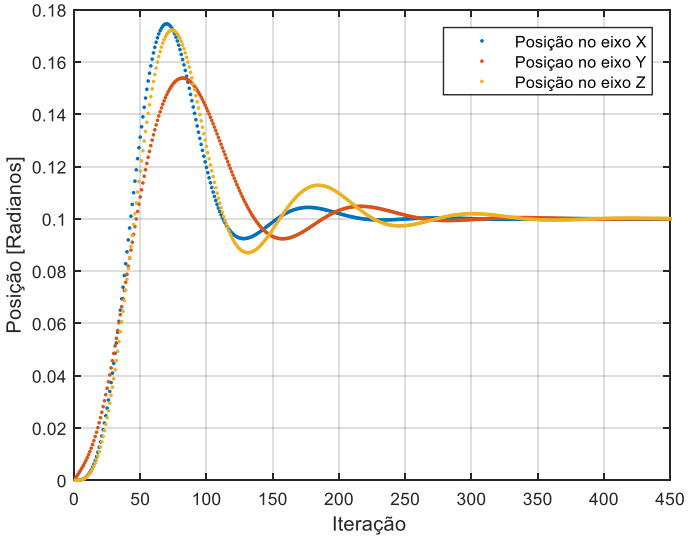
Figura 5.8 - Sinais de controle sem falha x Sinais de controle com falha.



Fonte: Produção do Autor.

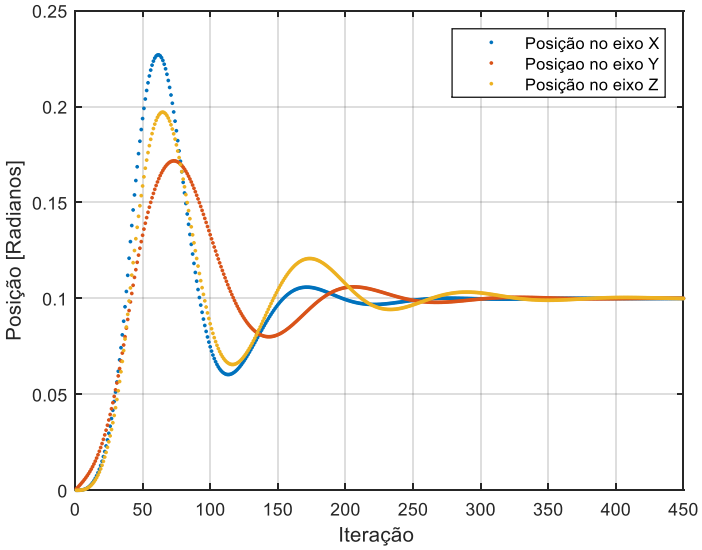
Na Figura 5.9 e na Figura 5.10, são apresentadas, respectivamente, a resposta da planta quando não há nenhuma falha inserida no sistema, e quando a falha, que representa a perda de dados entre o módulo principal e o RDC, é inserida no sistema SCA com a planta executada no módulo principal. A falha simulada demonstra que a perda de dados na comunicação degrada o desempenho (%M_p) do sistema, e que, em casos mais graves, pode causar instabilidade no mesmo, como demonstrado em Oliveira Júnior (2010).

Figura 5.9 - Resposta da planta - Sem falha.



Fonte: Produção do Autor.

Figura 5.10 - Resposta da planta - Com falha.

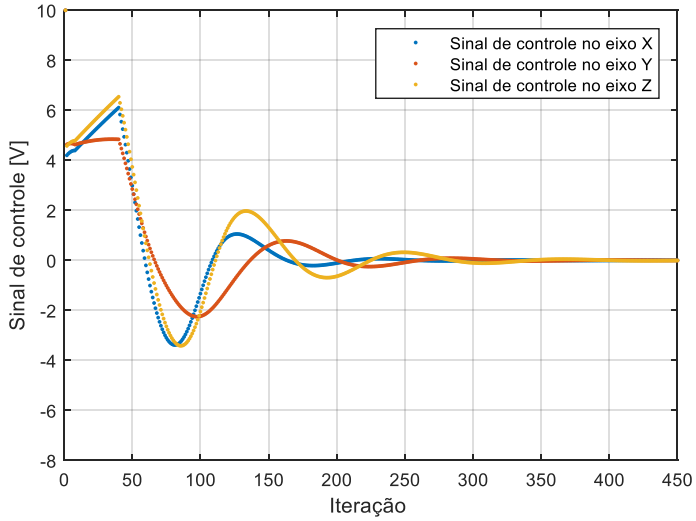


Fonte: Produção do Autor.

5.1.2.2. Análise *dead zero*

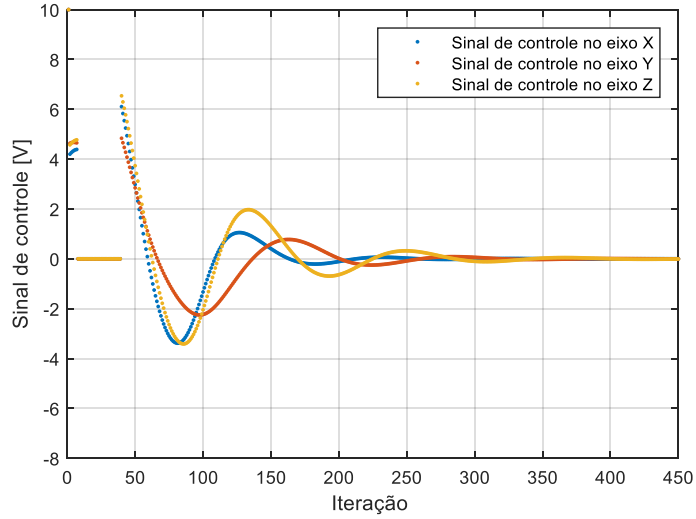
Na análise *dead zero*, enquanto os sinais de controle são impedidos de serem atualizados, i.e., desde o momento que ocorre a falha de comunicação, a planta não recebe sinal de controle, ou seja, recebe zero, Figura 5.12. No gráfico da Figura 5.11 é possível observar os sinais de controle enviados.

Figura 5.11 - Sinais de controle enviados.



Fonte: Produção do Autor.

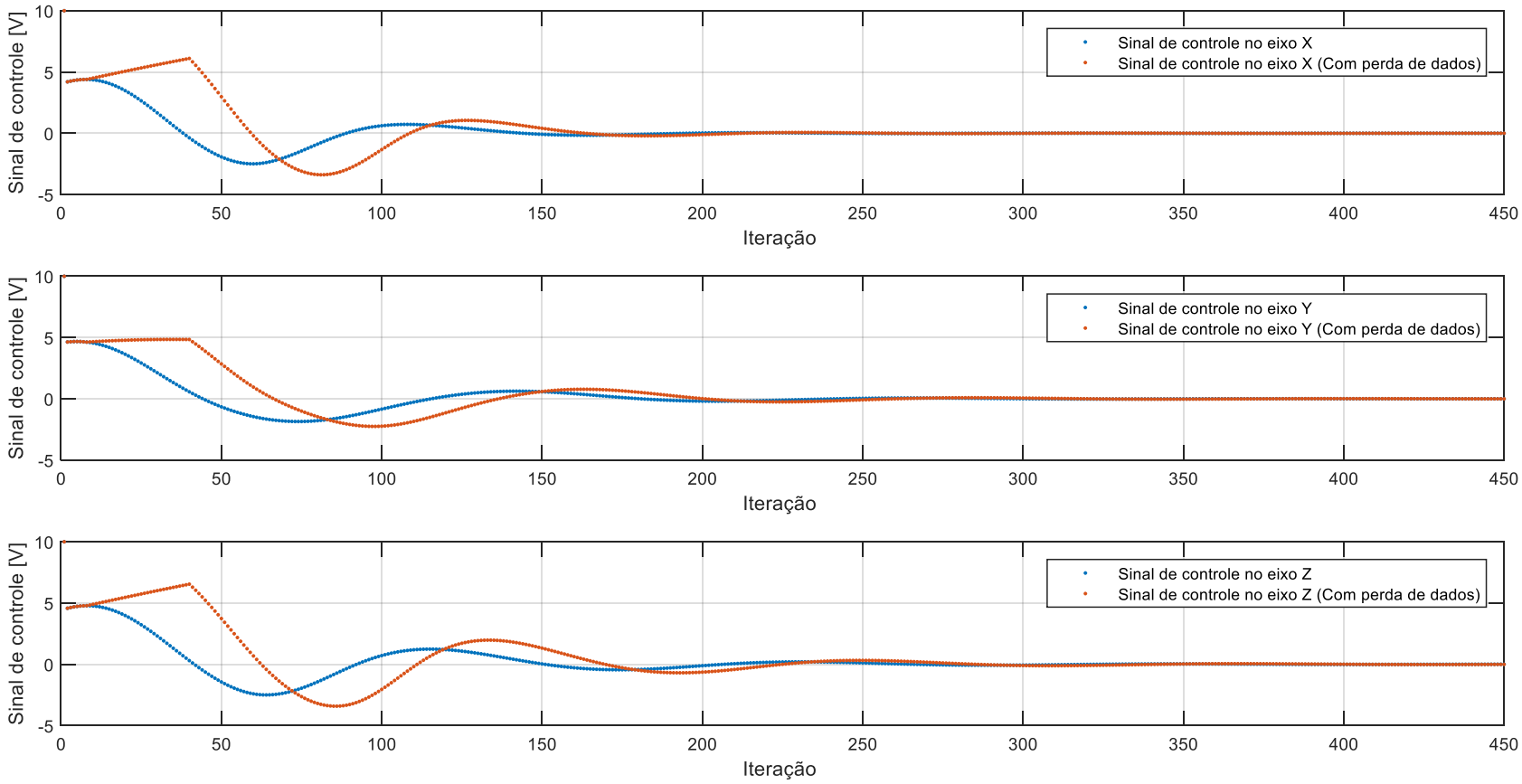
Figura 5.12 - Sinais de controle percebidos.



Fonte: Produção do Autor.

A Figura 5.13 permite comparar o sinal de controle em cada eixo, sem e com a falha, respectivamente

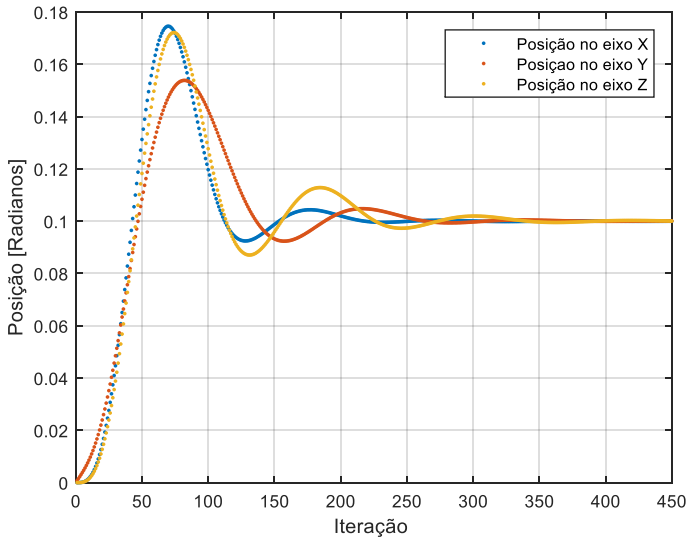
Figura 5.13 - Sinais de controle sem falha x Sinais de controle com falha.



Fonte: Produção do Autor.

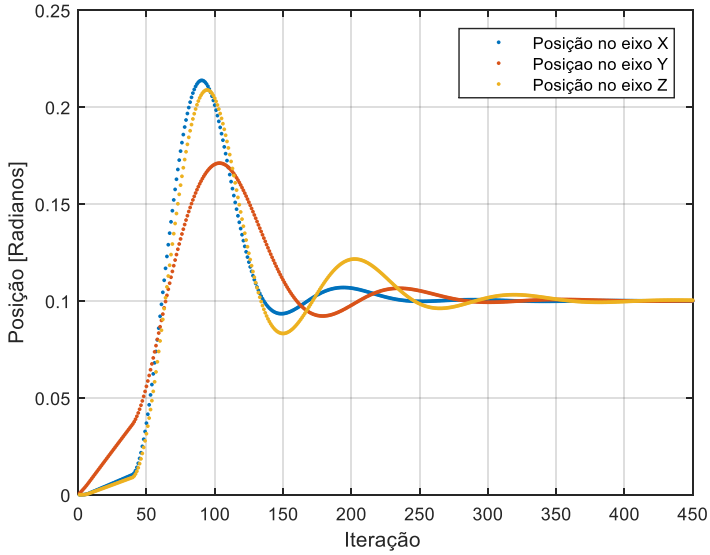
Na Figura 5.14 e na Figura 5.15 são apresentadas, respectivamente, a resposta da planta quando não há nenhuma falha inserida no sistema, e quando a falha, que representa a perda de dados entre o módulo principal e o RDC, é inserida no sistema SCA com planta executada no módulo principal. A falha simulada demonstra que a perda de dados na comunicação degrada o desempenho (%M_p) do sistema, e que, em casos mais graves, pode causar instabilidade no mesmo, como demonstrado em Oliveira Júnior (2010).

Figura 5.14 - Resposta da planta - Sem falha.



Fonte: Produção do Autor.

Figura 5.15 - Resposta da planta - Com falha.

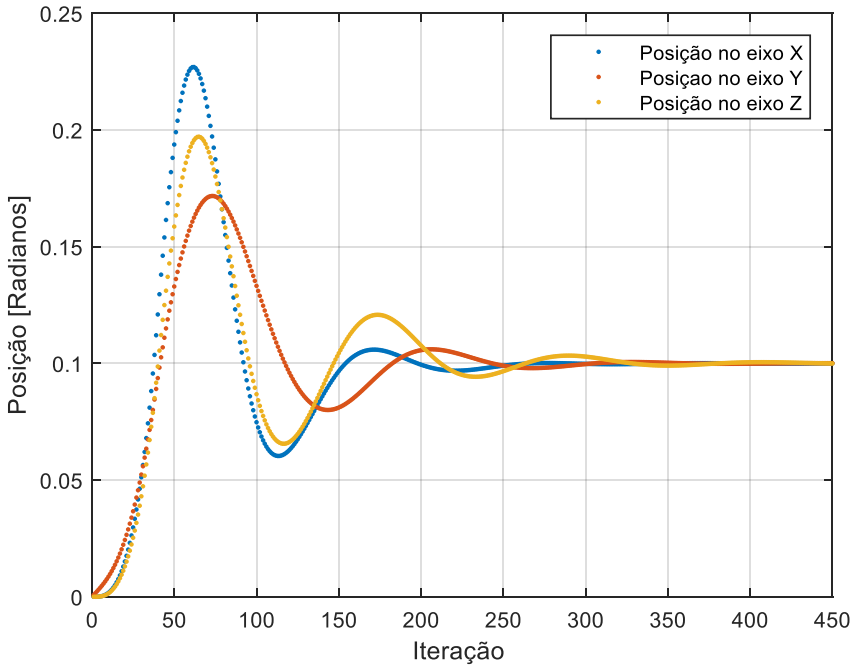


Fonte: Produção do Autor.

5.1.2.3. Análise *dead reckoning* x Análise *dead zero*

Ao comparar a resposta da planta nos dois casos analisados, nota-se que: quando a planta recebe e mantém o último valor, maior que zero, de sinal de controle imediatamente anterior à ocorrência da falha, durante a falha de comunicação (desde o dado 22 até o dado 117- desde a oitava iteração até a trigésima nona iteração), a posição da mesma cresce ao quadrado do tempo e sua velocidade cresce linearmente com o tempo, Figura 5.16.

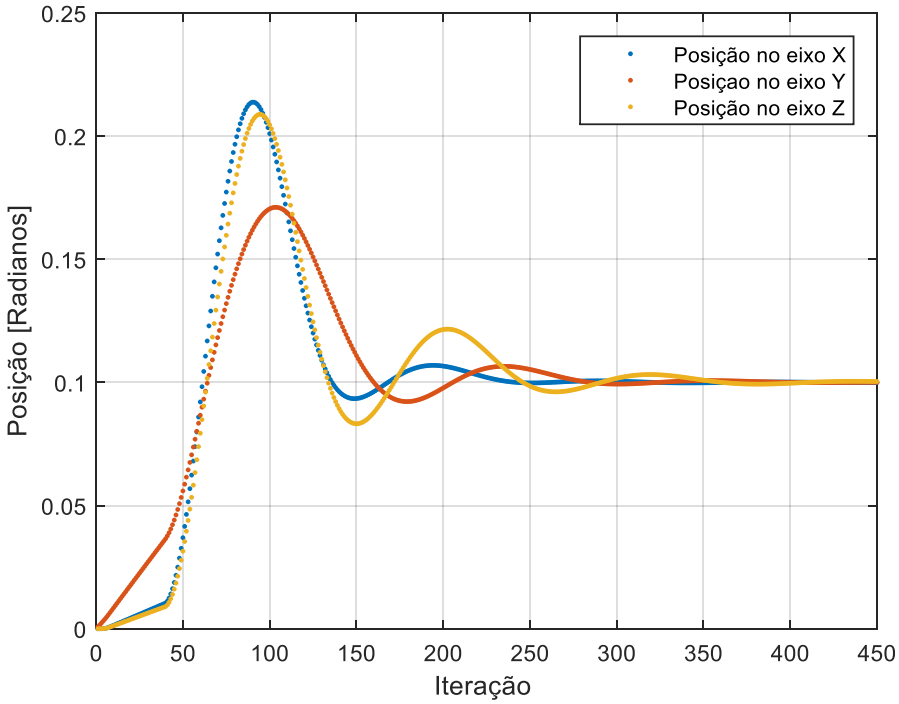
Figura 5.16 - Resposta da planta - *Dead reckoning.*



Fonte: Produção do Autor.

Enquanto, quando a planta recebe nenhum sinal de controle, ou seja, valor zero durante a falha de comunicação (desde o dado 22 até o dado 117- desde a oitava iteração até a trigésima nona iteração), a posição da mesma cresce linearmente com o tempo e sua velocidade é constante, maior que zero, Figura 5.17.

Figura 5.17 - Resposta da planta - *Dead zero.*



Fonte: Produção do Autor.

Apesar do *dead reckoning* ser bastante utilizado e conhecido como uma boa solução, no particular intervalo de tempo do sistema apresentado neste trabalho isso não acontece. O desempenho do sistema é pior ao utilizar o *dead reckoning* do que ao utilizar o *dead zero* porque o controle mantido, a velocidade e a posição têm o mesmo sinal durante aquele intervalo de tempo.

5.2. Implementação 1 - Caso de teste B

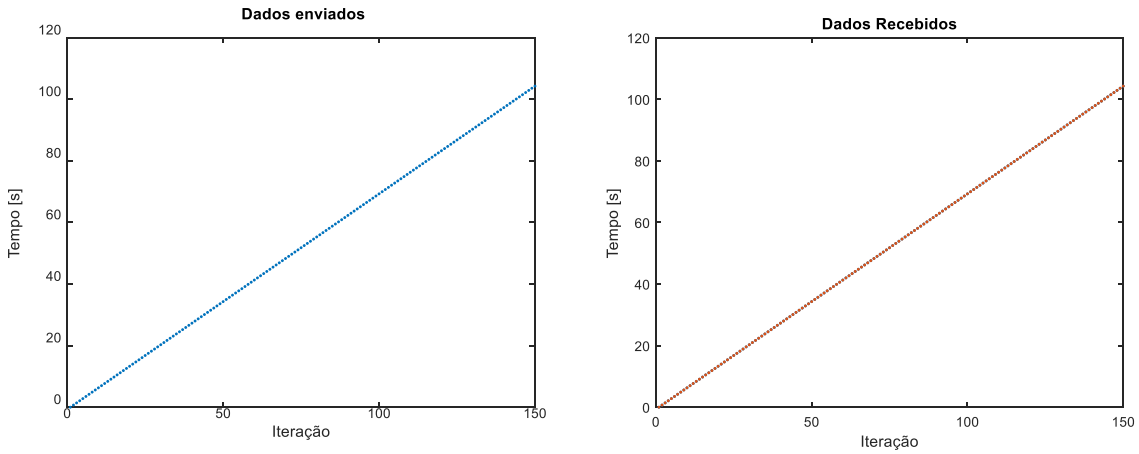
Nesta implementação o módulo principal envia para o RDC a saída de uma função do sistema operacional, que retorna o número de pulsos (*ticks)* do relógio decorridos desde que o sistema é inicializado. O número é obtido a cada iteração da partição *"pidsca"*, e então enviado para o RDC. Essa implementação, tem um total de 150 dados enviados, na qual as mensagens enviadas são organizadas em fila e lidas em ordem (FIFO).

5.2.1. Verificação lógica dos dados

A verificação lógica dos dados é realizada por meio da comparação entre os dados enviados pelo módulo principal e recebidos pelo RDC. É verificado se os dados são recebidos e se a sequência de envio é preservada no recebimento.

Na Figura 5.18, é possível notar que a ordem dos dados enviados é preservada, pois o dado enviado é sempre um valor crescente de tempo.

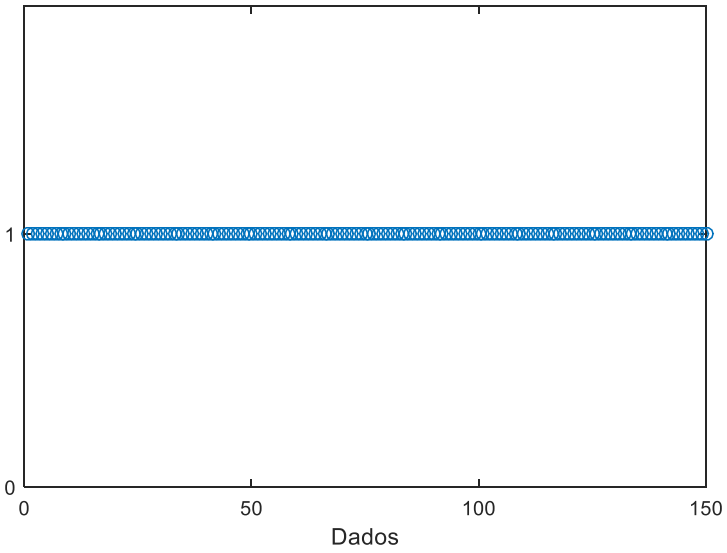
Figura 5.18 - Dados enviados x Dados recebidos.



Fonte: Produção do Autor.

Ao se fazer a verificação da entrega dos dados recebidos, é possível observar na Figura 5.19 que todos os dados são recebidos pelo RDC, na qual o número 1 representa os dados recebidos, e o número 0 representa os dados perdidos.

Figura 5.19 - Análise dos dados enviados.



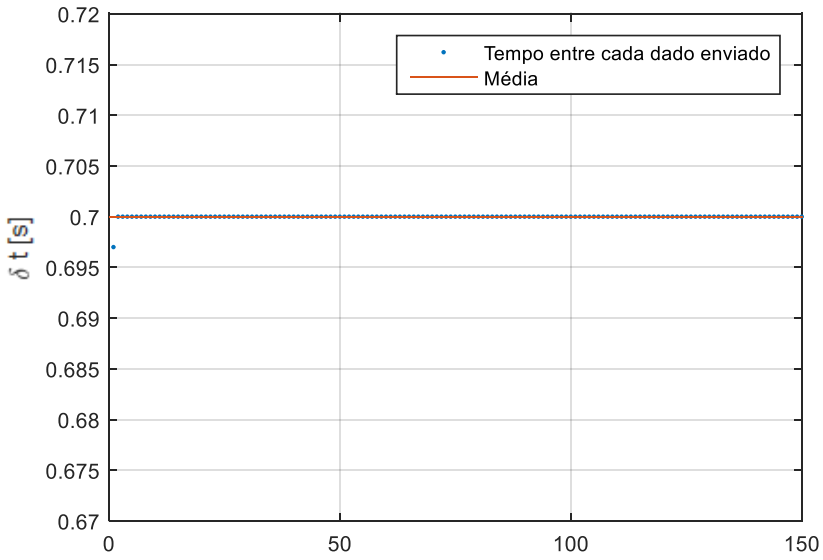
Fonte: Produção do Autor.

5.2.2. Análise temporal dos dados enviados

A análise temporal dos dados enviados pelo módulo principal é realizada por meio da diferença de tempo de envio entre cada iteração. É analisado o comportamento do sistema ao enviar os dados de acordo com sua periodicidade.

No módulo principal, cada partição é executada durante 0.1s com período de 0.7s. Então é esperado que a função do sistema operacional, que retorna o número de pulsos do relógio decorridos desde que o sistema é inicializado, retorne um valor crescente e múltiplo de 0.7. Logo é esperado que a diferença de tempo entre uma iteração e outra seja de 0.7s. A Figura 5.20 apresenta a diferença de tempo entre cada dado enviado.

Figura 5.20 - Diferença de tempo entre cada iteração.



Fonte: Produção do Autor.

Nota-se que a diferença de 0.7s é confirmada para todos os dados, exceto para o primeiro e o segundo. A diferença entre o primeiro e o segundo dado é de 0.697s. Isto ocorre provavelmente devido às rotinas de inicialização do sistema. Essa diferença não é significativa em relação à média. Isto é denotado pelo baixo desvio padrão conforme o valor apresentado na Tabela 5.2. Portanto, a primeira diferença pode ser considerada um *outlier,* i.e., ponto fora da curva.

Tabela 5.2 - Média, Desvio Padrão e Moda.

	Média	**Desvio Padrão**	**Moda**
Diferença entre cada iteração Tempo de envio	0.7	2.4577×10^{-4}	0.7

Fonte: Produção do Autor.

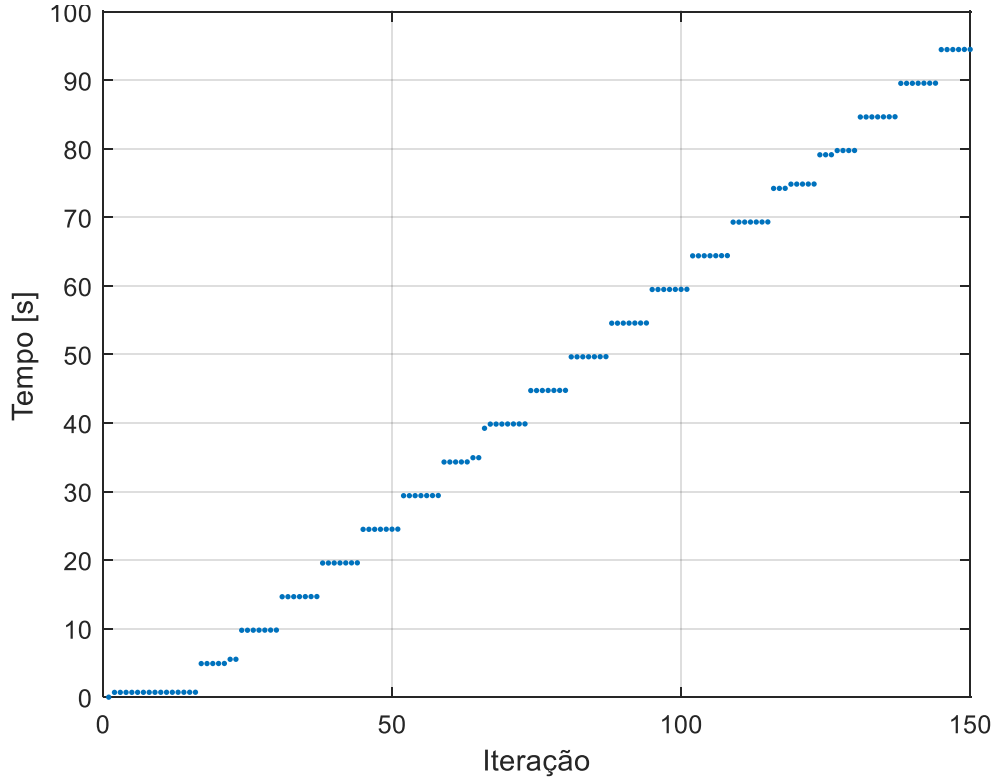
O resultado apresentado na Tabela 5.2 indica o alto nível de determinismo do sistema do módulo principal.

5.2.3. Análise temporal dos dados recebidos

A análise temporal das mensagens recebidas pelo RDC é realizada por meio da diferença de tempo de recebimento entre cada iteração. É analisado o comportamento do sistema ao receber os dados pela rede.

No RDC, o tempo de recebimento do primeiro dado é variável. Isso acontece devido à utilização da função própria do Arduino, que retorna o número de milissegundos passados desde que a placa começa a executar o programa. Para se fazer a análise do tempo em que cada dado é recebido, é preciso ter o tempo inicial definido. Considera-se então o tempo inicial de recebimento como zero segundos, logo é necessário fazer a correção do tempo de recebimento de todos os dados. Então desconta-se o tempo de recebimento do primeiro dado do tempo de recebimento de todos os dados. A Figura 5.21 a seguir apresenta o tempo, em segundos, em que cada dado é recebido após essa correção.

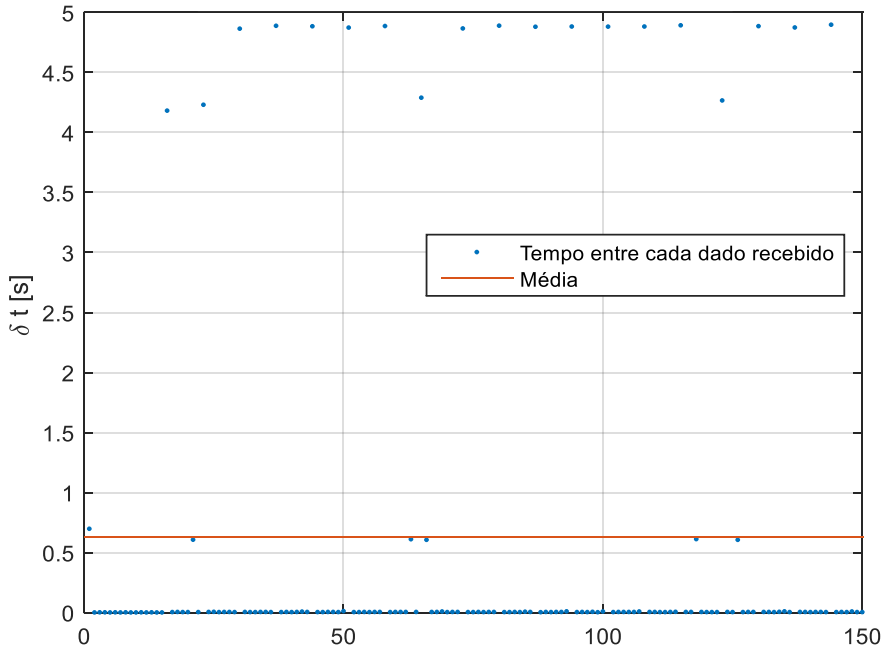
Figura 5.21 - Tempo de recebimento.



Fonte: Produção do Autor.

Ao fazer a diferença do tempo de recebimento entre cada iteração, Figura 5.22, é possível observar que existem faixas de tempo para o recebimento dos dados.

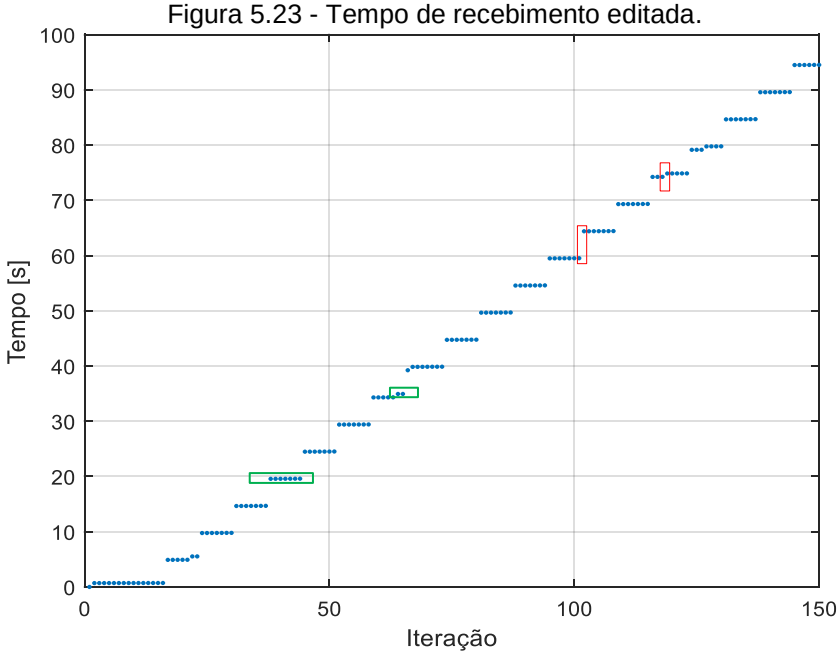
Figura 5.22 - Diferença de tempo entre cada iteração.



Fonte: Produção do Autor.

A Figura 5.23, que é a Figura 5.21 editada, mostra em quais momentos essas faixas podem ser observadas. Destacada em vermelho, tem-se a faixa que representa a maior diferença de tempo de recebimento, (Δt 1), entre cada dado recebido e em verde, a menor diferença de tempo de recebimento, (Δt 2).

Figura 5.23 - Tempo de recebimento editada.



Fonte: Produção do Autor.

O RDC leva um tempo, desconhecido e variável para perceber a disponibilidade dos dados na rede para recebimento. Quando percebe, já existem dados acumulados. Então enquanto há dados disponíveis para recebimento e perceptíveis ao RDC, o RDC recebe os dados, o que gera uma diferença quase nula entre o tempo de recebimento da maioria dos dados.

A Tabela 5.3 a seguir, apresenta o desvio padrão e a moda da diferença de tempo de recebimento entre cada dado. O desvio padrão relativamente alto, é esperado devido à faixa de maior diferença do tempo de recebimento. E a moda, é compatível com a existência da faixa de menor diferença do tempo de recebimento, que é mais representativa.

Tabela 5.3 - Desvio Padrão e Moda.

	Desvio Padrão	**Moda**
Diferença entre cada iteração Tempo de recebimento	1.5868	0.0050

Fonte: Produção do Autor.

A Tabela 5.4 a seguir, apresenta a média, o desvio padrão, a moda, o valor mínimo e o valor máximo da faixa de maior diferença do tempo de recebimento dos dados, (Δt 1), assim como a quantidade de ocorrências da mesma.

Tabela 5.4 - Média, Desvio Padrão, Moda, Quantidade, Valor Mínimo e Máximo.

	Média	**Desvio Padrão**	**Moda**
Diferença entre cada iteração - Faixa de maior diferença do tempo de recebimento Δt 1	3.7579	1.8129	4.8830
	Quantidade	**Valor Mínimo**	**Valor Máximo**
	25	0.6060	4.8980

Fonte: Produção do Autor.

A Tabela 5.5 a seguir, apresenta a média, o desvio padrão, a moda, o valor mínimo e o valor máximo da faixa de menor diferença do tempo de recebimento dos dados, (Δt 2), assim como a quantidade de ocorrências da mesma.

Tabela 5.5 - Média, Desvio Padrão, Moda, Quantidade, Valor Mínimo e Máximo.

	Média	**Desvio Padrão**	**Moda**
Diferença entre cada iteração - Faixa de menor diferença do tempo de recebimento Δt 2	0.0046	0.0018	0.005
	Quantidade	**Valor Mínimo**	**Valor Máximo**
	124	0.001	0.0120

Fonte: Produção do Autor.

5.3. Implementação 2 - Caso de teste A

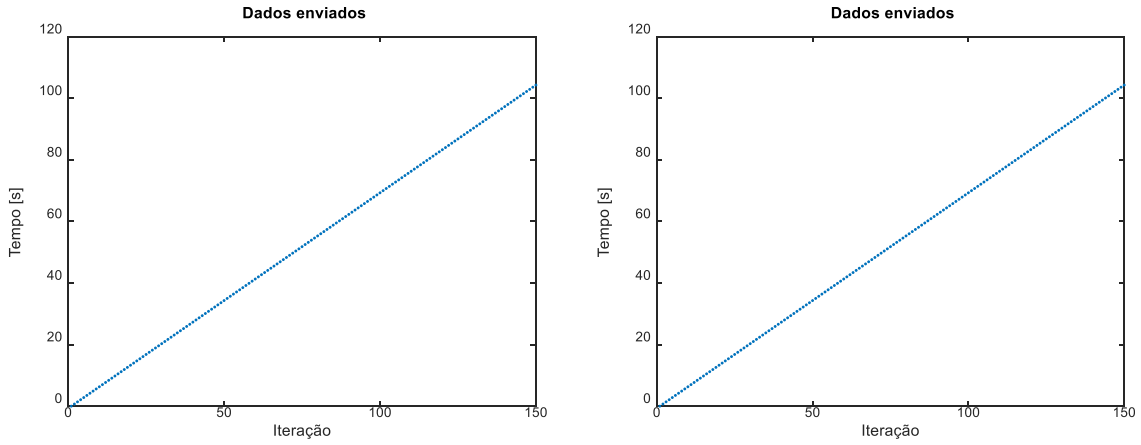
Nesta implementação o módulo principal envia para o RDC a saída de uma função do sistema operacional, que retorna o número de pulsos (*ticks)* do relógio decorridos desde que o sistema é inicializado. O número é obtido a cada iteração da partição *"pidsca"*, e então enviado para o RDC. Essa implementação, tem um total de 150 dados enviados, na qual as mensagens enviadas são organizadas em fila e lidas em ordem (FIFO). No RDC, a partição IOP, responsável pelo recebimento e gerenciamento dos dados, assim como a partição *receiver,* é executada durante 0.1s com período de 0.2s.

5.3.1. Verificação lógica dos dados

A verificação lógica dos dados é realizada por meio da comparação entre os dados enviados pelo módulo principal e recebidos pelo RDC. É verificado se os dados são recebidos e se a sequência de envio é preservada no recebimento.

Na Figura 5.24 a seguir, é possível notar que a ordem dos dados enviados é preservada, pois o dado enviado ao ser convertido para segundos, é sempre um valor crescente de tempo.

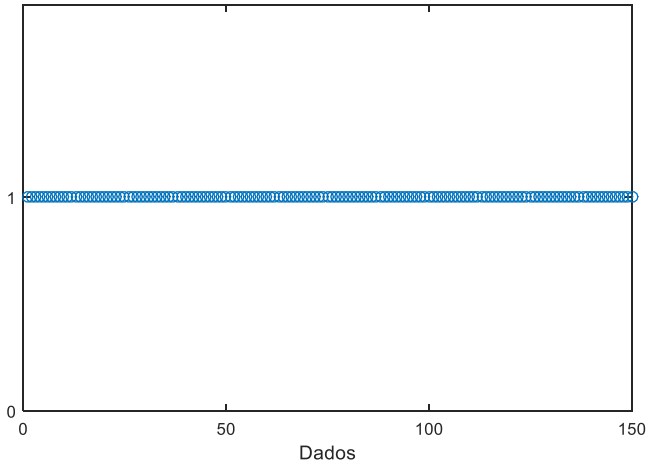
Figura 5.24 - Dados enviados x Dados recebidos.



Fonte: Produção do Autor.

Ao se fazer a análise dos dados recebidos, é possível perceber que todos os dados são recebidos pelo RDC. A Figura 5.25 apresenta esta análise, na qual o número 1 representa dados recebidos, e o número 0 representa dados perdidos.

Figura 5.25 - Análise dos dados enviados.



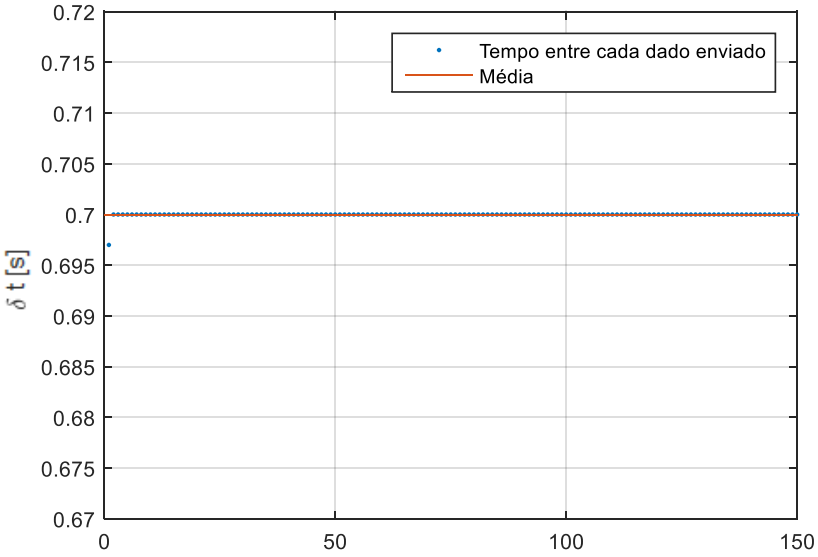
Fonte: Produção do Autor.

5.3.2. Análise temporal dos dados enviados

A análise temporal dos dados enviados pelo módulo principal é realizada por meio da diferença de tempo entre cada iteração. É analisado o comportamento do sistema ao enviar os dados de acordo com sua periodicidade.

No módulo principal, cada partição é executada durante 0.1s com período de 0.7s. Então é esperado que a função do sistema operacional, que retorna o número de pulsos do relógio decorridos desde que o sistema é inicializado, retorne um valor, que convertido para segundos seja crescente e múltiplo de 0.7. Logo é esperado que a diferença de tempo entre uma iteração e outra seja de 0.7s. A Figura 5.26 apresenta a diferença no tempo entre cada dado enviado.

Figura 5.26 - Diferença de tempo entre cada iteração.



Fonte: Produção do Autor.

Nota-se que a diferença de 0.7s é confirmada para todos os dados exceto para o primeiro e o segundo. A diferença entre o primeiro e o segundo dado é de 0.697s. Isto ocorre provavelmente devido às rotinas de inicialização do sistema. Essa diferença não é significativa em relação à média. Isto é denotado pelo baixo desvio padrão conforme o valor apresentado na Tabela 5.6. Portanto, a primeira diferença pode ser considerada um *outlier,* i.e., ponto fora da curva.

Tabela 5.6 - Média, Desvio Padrão e Moda.

	Média	**Desvio Padrão**	**Moda**
Diferença entre cada iteração Tempo de envio	0.7	2.4577×10^{-4}	0.7

Fonte: Produção do Autor.

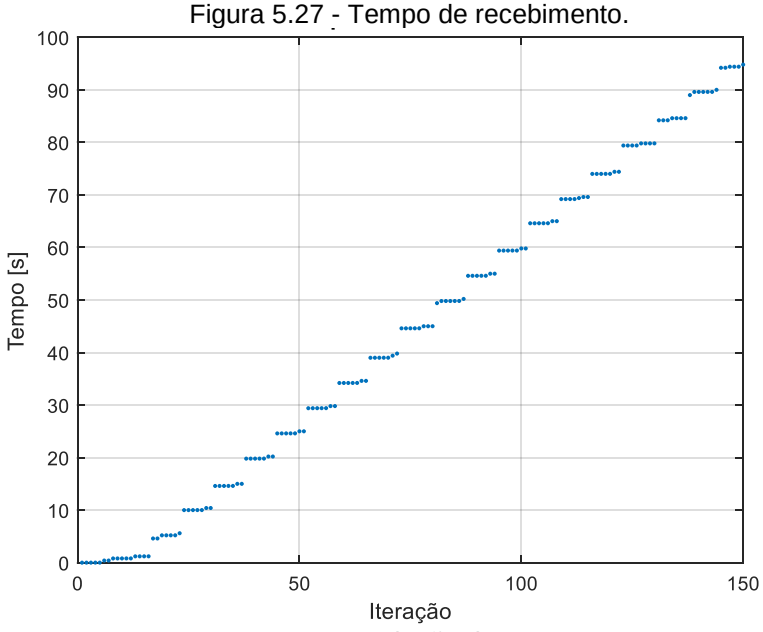
O resultado apresentado na tabela indica o alto nível de determinismo do sistema do módulo principal.

5.3.3. Análise temporal dos dados recebidos

A análise temporal das mensagens recebidas pelo RDC é realizada por meio da diferença de tempo de recebimento entre cada iteração. É analisado o comportamento do sistema ao receber os dados pela rede.

No RDC, o tempo de recebimento do primeiro dado é variável. Isso acontece devido à utilização da função do sistema operacional, que retorna o número de pulsos do relógio decorridos desde que o sistema é inicializado. Para se fazer a análise do tempo em que cada dado é recebido, é preciso ter o tempo inicial definido. Considera-se então o tempo inicial de recebimento como zero segundos, logo é necessário fazer a correção do tempo de recebimento de todos os dados. Então, desconta-se o tempo de recebimento do primeiro dado do tempo de recebimento de todos os dados. A Figura 5.18, a seguir, apresenta o tempo, em segundos, em que cada dado é recebido após essa correção.

Figura 5.27 - Tempo de recebimento.

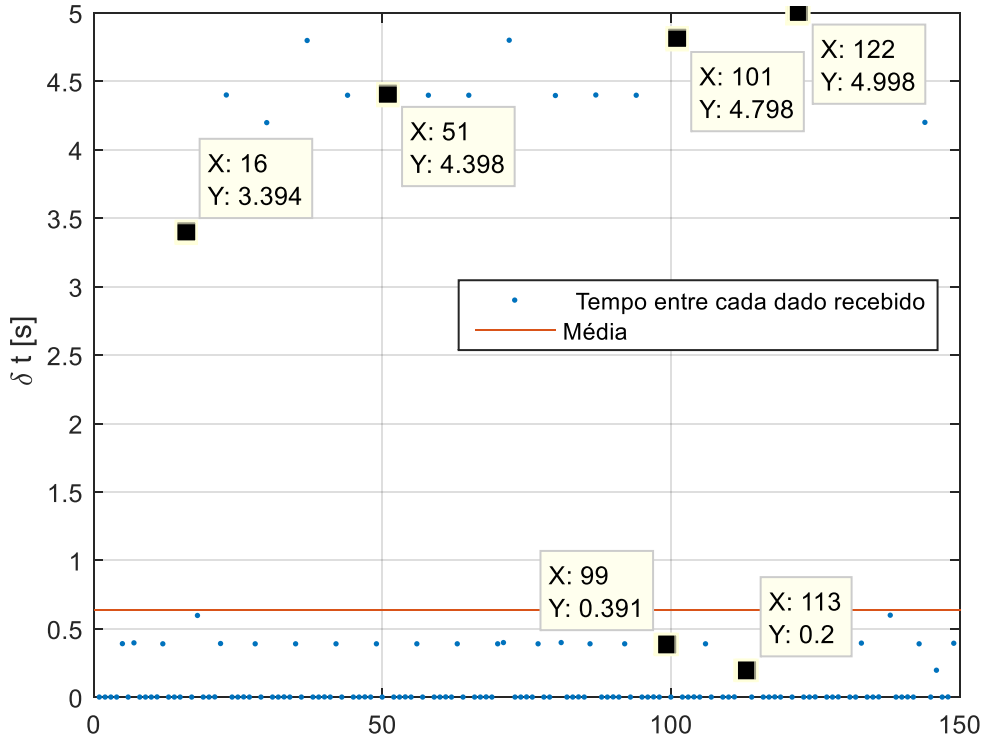


Fonte: Produção do Autor.

Na Figura 5.27 é possível observar que existem duas faixas de tempo para o recebimento dos dados, assim como na implementação 1- caso B, uma faixa que representa a maior diferença de tempo de recebimento, (Δt 1), e uma faixa que representa a menor diferença de tempo de recebimento, (Δt 2), entre cada dado

recebido. A Figura 5.28 a seguir apresenta a diferença de tempo entre cada iteração.

Figura 5.28 - Diferença de tempo entre cada iteração.



Fonte: Produção do Autor.

A cada iteração com período de 0,7s, a partição “pidsca” do módulo principal envia um dado; já o RDC pode receber *n* dados a cada iteração de 0,1s de duração e período de 0.2s. Nota-se, que os valores da diferença de tempo de recebimento entre um dado e outro são bem próximos à valores múltiplos do período do RDC. Essas flutuações ocorrem provavelmente devido às incertezas temporais do hardware.

O RDC leva um tempo, desconhecido e variável para perceber a disponibilidade dos dados na rede para recebimento. Quando percebe, já existem dados acumulados. Então é provável que enquanto há dados disponíveis para recebimento e perceptíveis ao RDC, o RDC os recebe, o que gera uma diferença quase nula entre o tempo de recebimento da maioria dos dados.

A Tabela 5.7, apresenta o desvio padrão e a moda da diferença de tempo de recebimento entre cada dado. O desvio padrão relativamente alto, é esperado devido à faixa de maior diferença do tempo de recebimento. E a moda, é compatível com a existência da faixa de menor diferença do tempo de recebimento, que é mais representativa.

Tabela 5.7 - Desvio Padrão e Moda.

	Desvio Padrão	**Moda**
Diferença entre cada iteração Tempo de recebimento	1.4594	0.0020

Fonte: Produção do Autor.

A Tabela 5.8 a seguir, apresenta a média, o desvio padrão, a moda, o valor mínimo e o valor máximo da faixa de maior diferença do tempo de recebimento dos dados, (Δt 1), assim como a quantidade de ocorrências da mesma.

Tabela 5.8 - Média, Desvio Padrão, Moda, Quantidade, Valor Mínimo e Máximo.

	Média	**Desvio Padrão**	**Moda**
Diferença entre cada iteração - Faixa de maior diferença do tempo de recebimento Δt 1	2.0122	2.0067	0.3910
	Quantidade	**Valor Mínimo**	**Valor Máximo**
	47	0.1930	4.9980

Fonte: Produção do Autor.

A Tabela 5.9 a seguir, apresenta a média, o desvio padrão, a moda, o valor mínimo e o valor máximo da faixa de menor diferença do tempo de recebimento dos dados, (Δt 2), assim como a quantidade de ocorrências da mesma.

Tabela 5.9 - Média, Desvio Padrão, Moda, Quantidade, Valor Mínimo e Máximo.

	Média	**Desvio Padrão**	**Moda**
Diferença entre cada iteração - Faixa de menor diferença do tempo de recebimento Δt 2	0.0022	0.000406	0.002
	Quantidade	**Valor Mínimo**	**Valor Máximo**
	102	0.002	0.003

Fonte: Produção do Autor.

5.4. Implementação 2 - Caso de teste B

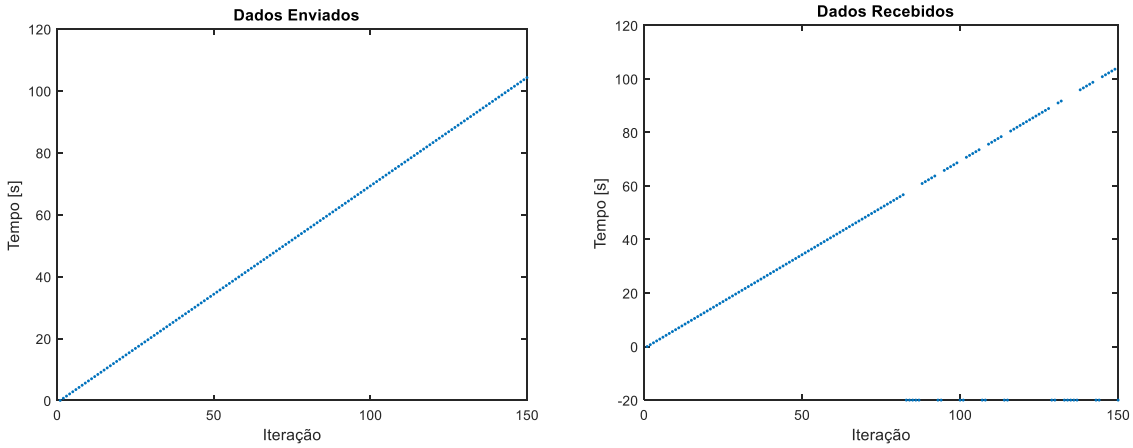
Nesta implementação o módulo principal envia para o RDC a saída de uma função do sistema operacional, que retorna o número de pulsos (*ticks)* do relógio decorridos desde que o sistema é inicializado. O número é obtido a cada iteração da partição *"pidsca"*, e então enviado para o RDC. Essa implementação, tem um total de 150 dados enviados, na qual as mensagens enviadas são organizadas em fila e lidas em ordem (FIFO). No RDC, a partição IOP2 e a partição *receiver,* possuem período de 0.7s e são executadas durante 0.1s.

5.4.1. Verificação lógica dos dados

A verificação lógica dos dados é realizada por meio da comparação entre os dados enviados pelo módulo principal e recebidos pelo RDC. É verificado se os dados são recebidos e se a sequência de envio é preservada no recebimento.

Na Figura 5.29, a seguir, é possível notar que a ordem dos dados enviados é preservada, pois o dado enviado ao ser convertido para segundos, é sempre um valor crescente de tempo. Nesta implementação alguns dados são perdidos durante a comunicação e para esses dados perdidos, atribui-se o número -20 para uma melhor visualização do fenômeno.

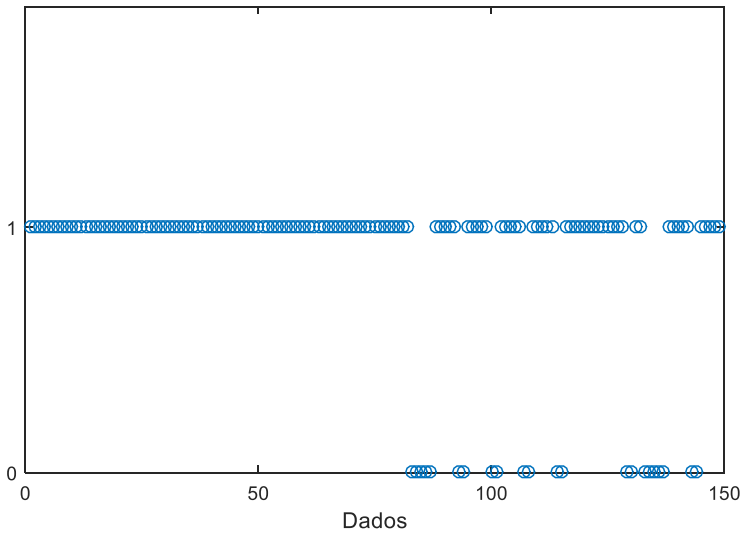
Figura 5.29 - Dados enviados x Dados recebidos.



Fonte: Produção do Autor.

Na Figura 5.30, é possivel perceber melhor essa perda de dados, na qual o número 1 representa dados recebidos, e o número 0 representa dados perdidos. Dos dados enviados, o RDC recebeu 84.6667%, (127 dados recebidos de 150 enviados).

Figura 5.30 - Análise dos dados enviados.



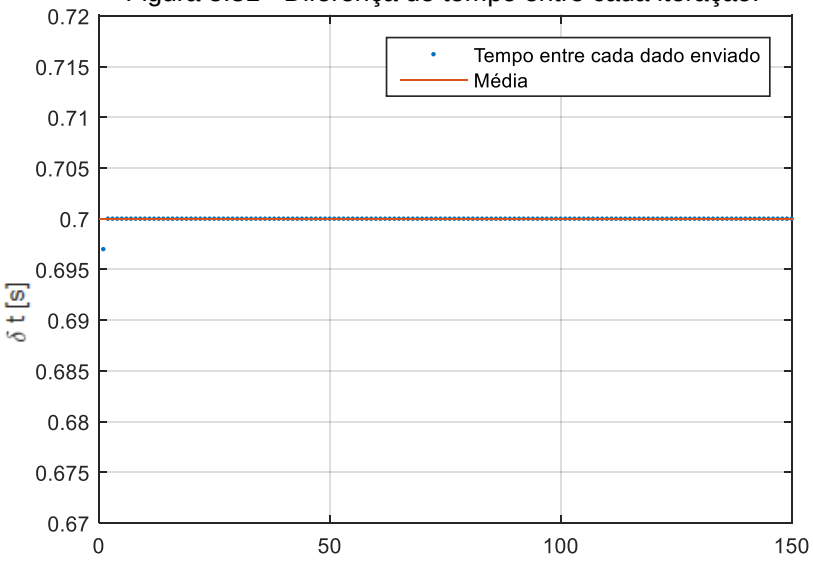
Fonte: Produção do Autor.

5.4.2. Análise temporal dos dados enviados

A análise temporal dos dados enviados pelo módulo principal é realizada por meio da diferença de tempo entre cada iteração. É analisado o comportamento do sistema ao enviar os dados de acordo com sua periodicidade.

No módulo principal, cada partição é executada durante 0.1s com período de 0.7s. Então é esperado que a função do sistema operacional, que retorna o número de pulsos do relógio decorridos desde que o sistema é inicializado, retorne um valor, que convertido para segundos seja crescente e múltiplo de 0.7. Logo é esperado que a diferença de tempo entre uma iteração e outra seja de 0.7s. A Figura 5.31 apresenta a diferença no tempo entre cada dado enviado.

Figura 5.31 - Diferença de tempo entre cada iteração.



Fonte: Produção do Autor.

Nota-se que a diferença de 0.7s é confirmada para todos os dados exceto para o primeiro e o segundo. A diferença entre o primeiro e o segundo dado é de 0.697s. Isto ocorre provavelmente devido às rotinas da inicialização do sistema. Essa diferença não é significativa em relação à média. Isto é denotado pelo baixo desvio padrão conforme o valor apresentado na Tabela 5.10. Portanto, a primeira diferença pode ser considerada um *outlier,* i.e., ponto fora da curva.

Tabela 5.10 - Média, Desvio Padrão e Moda.

	Média	**Desvio Padrão**	**Moda**
Diferença entre cada iteração Tempo de envio	0.7	2.4577×10^{-4}	0.7

Fonte: Produção do Autor.

O resultado apresentado na Tabela 5.10 indica o alto nível de determinismo do sistema do módulo principal.

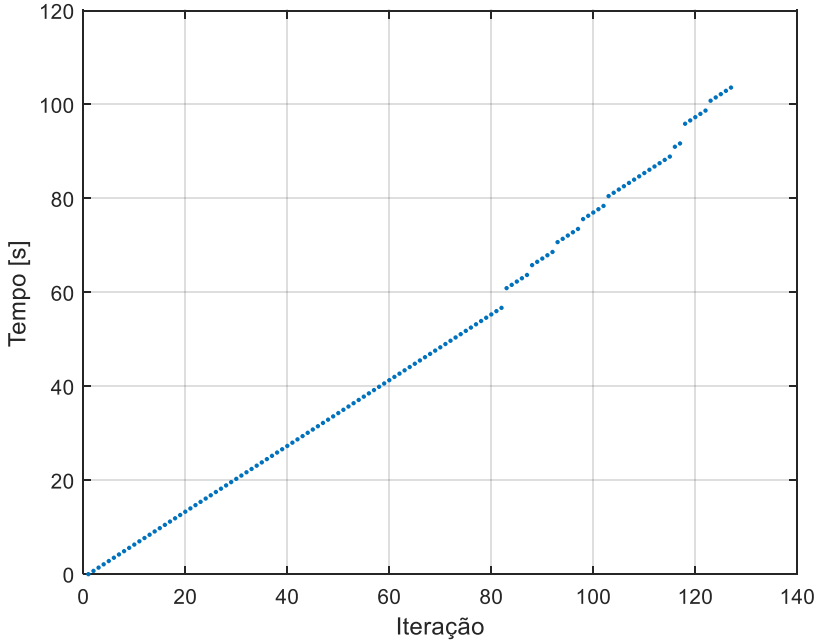
5.4.3. Análise temporal dos dados recebidos

A análise temporal das mensagens recebidas pelo RDC é realizada por meio da diferença de tempo de recebimento entre cada iteração. É analisado o comportamento do sistema ao receber os dados pela rede.

O conteúdo das mensagens recebidas é o tempo em que as mesmas são enviadas, isso possibilita a análise do tempo de envio dos dados recebidos quando há falha na comunicação e perda de dados.

Nesta implementação, não há o recebimento de todos os dados enviados pelo módulo principal. São recebidos 127 dados de 150 enviados. Então na Figura 5.32, é possível observar a sequência de dados, convertidos para segundos, no qual o recebimento é bem-sucedido.

Figura 5.32 - Tempo de envio dos dados recebidos.

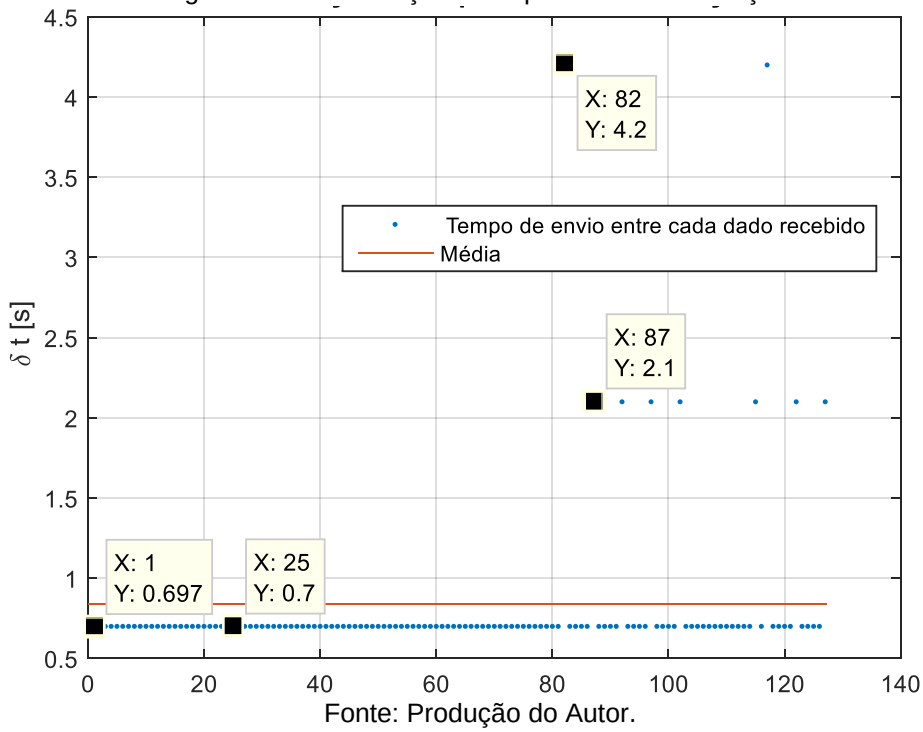


Fonte: Produção do Autor.

É provável que a perda de alguns dos dados enviados neste caso (caso 2b) aconteça devido à frequência que o RDC nota a existência de dados no canal/porta de comunicação e efetua o recebimento do mesmo. Como essa frequência é menor que a frequência da implementação anterior (caso 2a), isso sugere que ele não consiga notar a existência de dados no canal/porta de comunicação e efetuar o recebimento dos dados em tempo hábil, ou seja, retirar as mensagens/os dados do canal/porta, que faz com que o mesmo não tenha espaço disponível para receber novas mensagens. Como não há espaço no canal/porta, as mensagens enviadas pela partição “pidsca” para a IOP1 são perdidas quando a mesma tenta enviar para a rede. O reenvio dos dados perdidos não é solicitado pela rede, pois o protocolo de comunicação utilizado é o UDP.

Quando há espaço novamente no canal/porta, as mensagens enviadas no mesmo, se referem a uma outra chamada da função do sistema operacional utilizada. Como o conteúdo das mensagens enviadas é o tempo em que as mesmas são enviadas, é possível observar uma diferença de tempo de envio maior entre os dados recebidos. Porém, essa diferença é sempre múltipla de 0.7s, que corresponde ao agendamento do módulo principal, responsável pelo envio dos dados, como mostra a Figura 5.33.

Figura 5.33 - Diferença de tempo entre cada iteração.



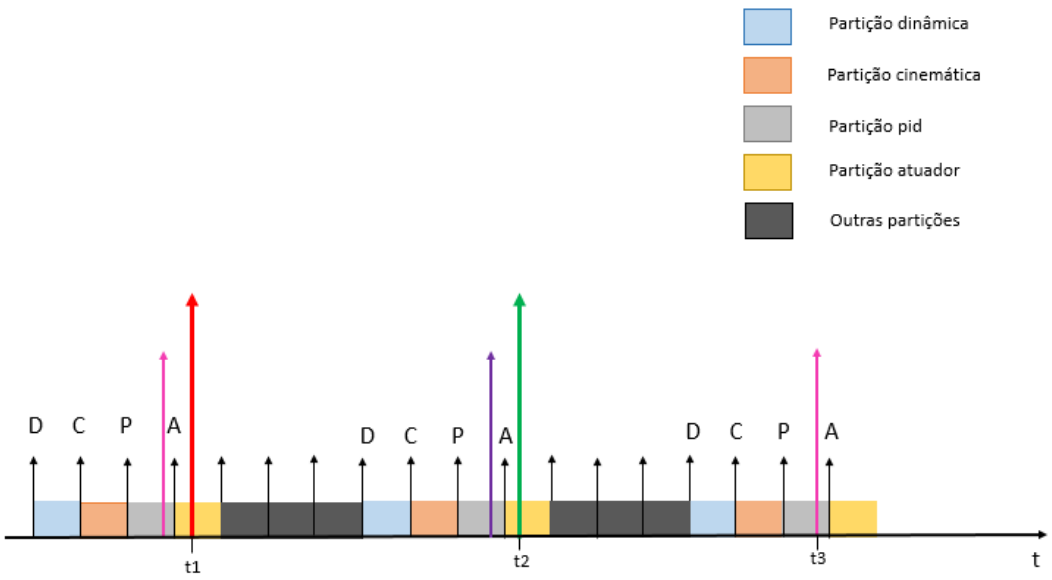
Fonte: Produção do Autor.

Como analisado na seção anterior, a diferença de tempo de envio entre a primeira e a segunda iteração, 0.697s, pode ser considerada um ponto fora da curva.

A perda de dados acontece provavelmente devido à falta de espaço no canal/porta, ocasionada pela frequência que o RDC efetua o recebimento dos dados. A Figura 5.34, a seguir, apresenta uma escala de tempo para observar a sequência de acontecimentos que, provavelmente, leva à ocorrência desse fenômeno. Para facilitar a observação, assume-se que o dado é enviado

diretamente para a rede pela partição “pidsca”, desconsiderando a existência da partição IOP1.

Figura 5.34 - Escala de tempo.



Fonte: Produção do Autor.

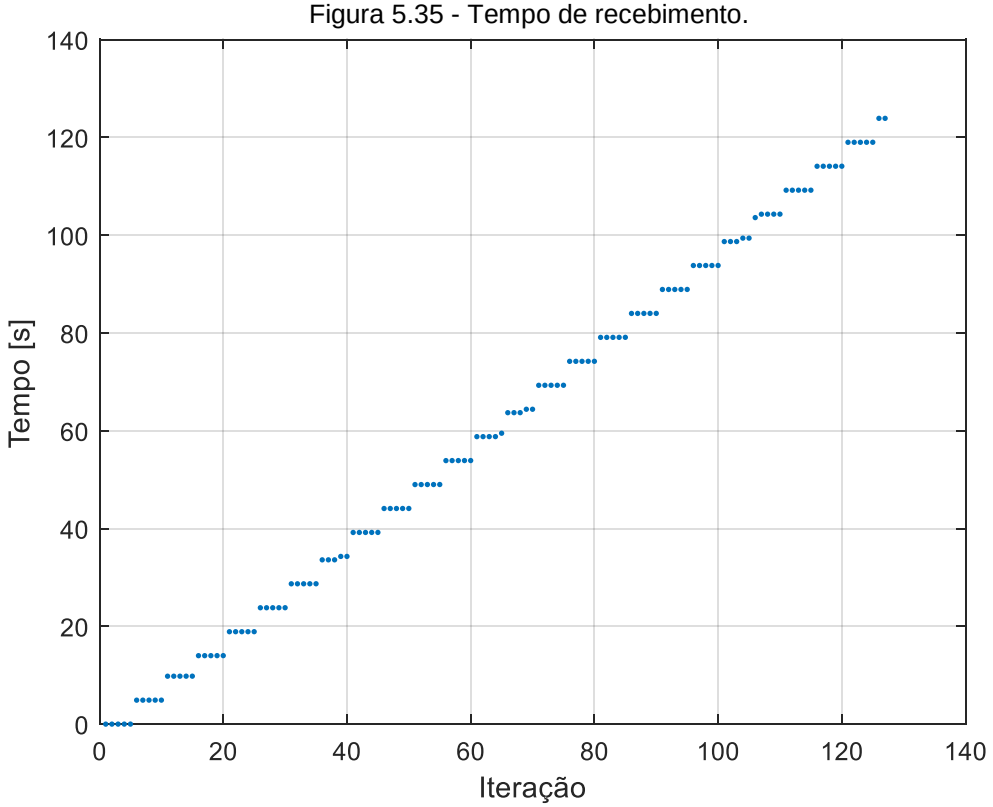
Após a mensagem ser enviada pela partição *“pidsca”*, primeira *flag* rosa, o canal/porta passa a não ter mais espaço, *flag* vermelha no instante t1. Quando a partição *“pidsca*” envia uma nova mensagem, a mesma é perdida, *flag* roxa, pois o canal/porta ainda está cheio. No instante t2, é liberado espaço no canal/porta, *flag* verde. No instante t3, quando a partição *“pidsca*” envia uma nova mensagem, a mesma é enviada para a rede, *flag* rosa, pois há espaço no canal/porta de comunicação.

Ao fazer a análise do tempo em que cada dado é recebido, é preciso ter o tempo inicial definido, pois no RDC, o tempo de recebimento do primeiro dado é variável. Isso acontece devido à utilização da função do sistema operacional, que retorna o número de pulsos do relógio decorridos desde que o sistema é inicializado.

Considera-se então o tempo inicial de recebimento como zero segundos; sendo assim, é necessário fazer a correção do tempo de recebimento de todos os

dados. Então, desconta-se o tempo de recebimento do primeiro dado do tempo de recebimento de todos os dados. A Figura 5.35, a seguir, apresenta o tempo, em segundos, em que cada dado é recebido após essa correção.

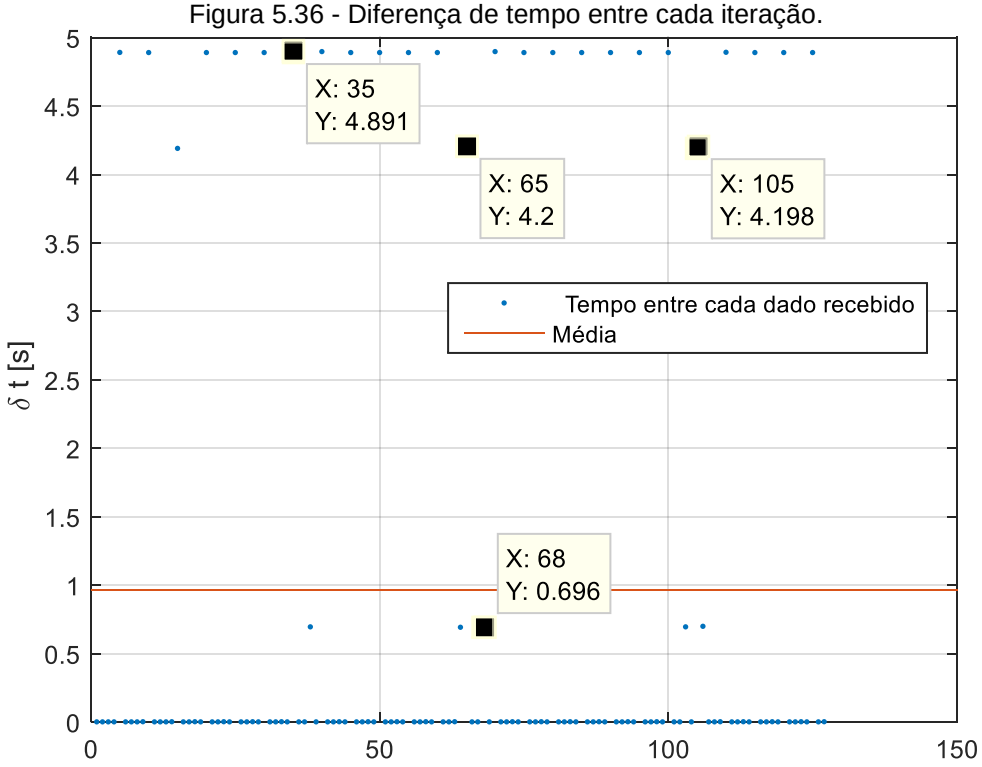
Figura 5.35 - Tempo de recebimento.



Fonte: Produção do Autor.

Na Figura 5.35, é possível observar que existem duas faixas de tempo para o recebimento dos dados, assim como nas implementações anteriores, uma faixa que representa a maior diferença de tempo de recebimento, (Δt 1), e uma faixa que representa a menor diferença de tempo de recebimento, (Δt 2) entre cada dado recebido. A Figura 5.36, a seguir, apresenta a diferença de tempo entre cada iteração.

Figura 5.36 - Diferença de tempo entre cada iteração.



Fonte: Produção do Autor.

A cada iteração com período de 0,7s, a partição “pidsca” do módulo principal envia um dado; já o RDC pode receber *n* dados a cada iteração de 0,1s de duração e período de 0.2s. Nota-se, que os valores da diferença de tempo de recebimento entre um dado e outro são bem próximos à valores múltiplos do período do RDC. Essas flutuações ocorrem provavelmente devido às incertezas temporais do hardware.

O RDC leva um tempo, desconhecido e variável para perceber a disponibilidade dos dados na rede para recebimento. Quando percebe, já existem dados acumulados. Então é provável que, enquanto há dados disponíveis para recebimento e perceptíveis ao RDC, o RDC os recebe, o que gera uma diferença quase nula entre o tempo de recebimento da maioria dos dados.

A Tabela 5.11, apresenta o desvio padrão e a moda da diferença de tempo de recebimento entre cada dado. O desvio padrão, relativamente alto, é esperado devido à faixa de maior diferença do tempo de recebimento. E a moda, é

compatível com a existência da faixa de menor diferença do tempo de recebimento, que é mais representativa.

Tabela 5.11 - Desvio Padrão e Moda.

	Desvio Padrão	**Moda**
Diferença entre cada iteração Tempo de recebimento	1.9180	0.0020

Fonte: Produção do Autor.

A Tabela 5.12 a seguir, apresenta a média, o desvio padrão, a moda, o valor mínimo e o valor máximo da faixa de maior diferença do tempo de recebimento dos dados, (Δt 1), assim como a quantidade de ocorrências da mesma.

Tabela 5.12 - Média, Desvio Padrão, Moda, Quantidade, Valor Mínimo e Máximo.

	Média	**Desvio Padrão**	**Moda**
Diferença entre cada iteração - Faixa de maior diferença do tempo de recebimento Δt 1	4.1230	1.5728	4.8910
	Quantidade	**Valor Mínimo**	**Valor Máximo**
	30	0.6930	4.8980

Fonte: Produção do Autor.

A Tabela 5.13 a seguir, apresenta a média, o desvio padrão, a moda, o valor mínimo e o valor máximo da faixa de menor diferença do tempo de recebimento dos dados, (Δt 2), assim como a quantidade de ocorrências da mesma.

Tabela 5.13 - Média, Desvio Padrão, Moda, Quantidade, Valor Mínimo e Máximo.

	Média	**Desvio Padrão**	**Moda**
Diferença entre cada iteração - Faixa de menor diferença do tempo de recebimento Δt 2	0.0022	0.000415	0.002
	Quantidade	**Valor Mínimo**	**Valor Máximo**
	96	0.002	0.003

Fonte: Produção do Autor.

5.5. Implementação 2 - Caso de teste C

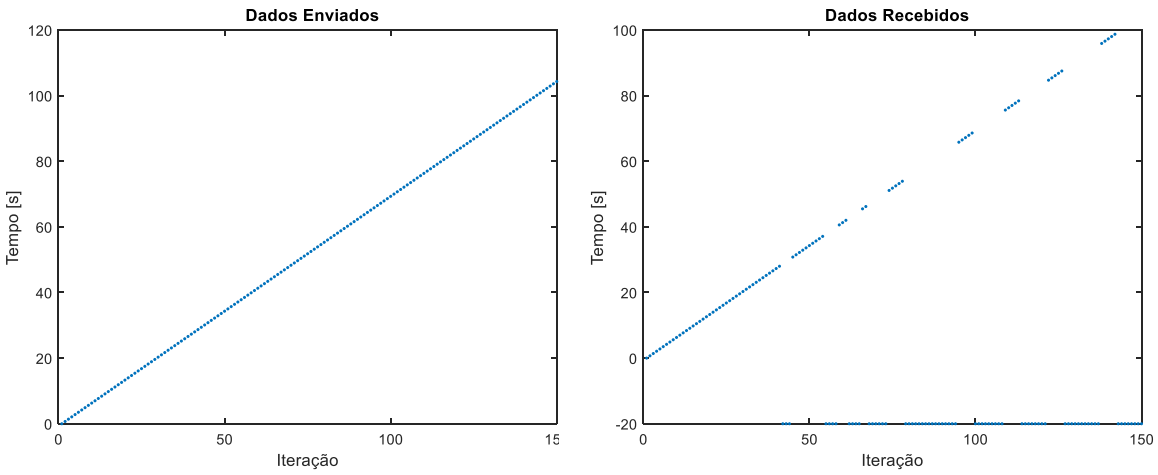
Nesta implementação, o módulo principal envia para o RDC a saída de uma função do sistema operacional, que retorna o número de pulsos (*ticks)* do relógio decorridos desde que o sistema é inicializado. O número é obtido a cada iteração da partição *"pidsca"*, e então enviado para o RDC. Essa implementação, tem um total de 150 dados enviados, na qual as mensagens enviadas são organizadas em fila e lidas em ordem (FIFO). No RDC, a partição IOP2 e a partição *receiver,* possuem período de 1.0s e são executadas durante 0.1s.

5.5.1. Verificação lógica dos dados

A verificação lógica dos dados é realizada por meio da comparação entre os dados enviados pelo módulo principal e recebidos pelo RDC. É verificado se os dados são recebidos e se a sequência de envio é preservada no recebimento.

Na Figura 5.37, a seguir, é possível notar que a ordem dos dados enviados é preservada, pois o dado enviado ao ser convertido para segundos, é sempre um valor crescente de tempo. Nesta implementação alguns dados são perdidos durante a comunicação e para esses dados perdidos, atribui-se o número -20 para uma melhor visualização do fenômeno.

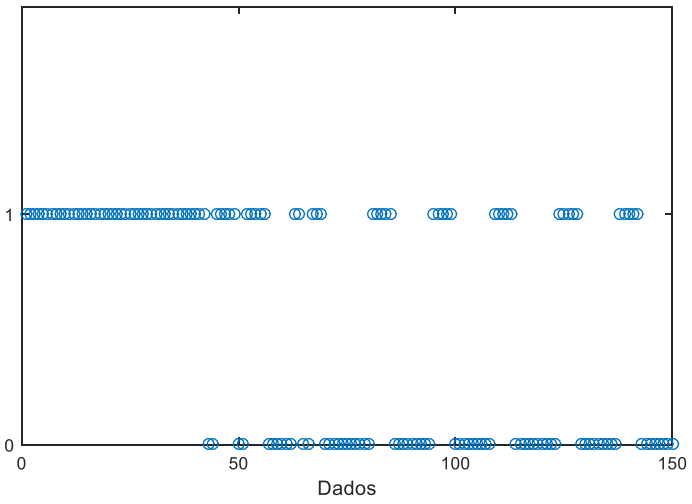
Figura 5.37 - Dados enviados x Dados recebidos.



Fonte: Produção do Autor.

Na Figura 5.38, a seguir, é possivel perceber melhor essa perda de dados, na qual na qual o número 1 representa dados recebidos, e o número 0 representa dados perdidos. Dos dados enviados, o RDC recebeu 54.6667%, (82 dados recebidos de 150 enviados).

Figura 5.38 - Análise dos dados enviados.



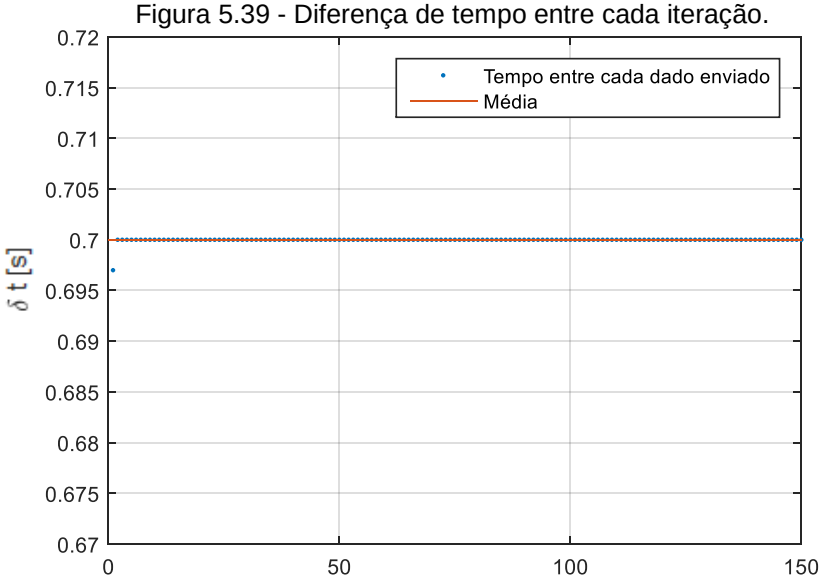
Fonte: Produção do Autor.

5.5.2. Análise temporal dos dados enviados

A análise temporal dos dados enviados pelo módulo principal é realizada por meio da diferença de tempo entre cada iteração. É analisado o comportamento do sistema ao enviar os dados de acordo com sua periodicidade.

No módulo principal, cada partição é executada durante 0.1s com período de 0.7s. Então é esperado que a função do sistema operacional, que retorna o número de pulsos do relógio decorridos desde que o sistema é inicializado, retorne um valor, que, convertido para segundos, seja crescente e múltiplo de 0.7. Logo é esperado que a diferença de tempo entre uma iteração e outra seja de 0.7s. A Figura 5.39 apresenta a diferença no tempo entre cada dado enviado.

Figura 5.39 - Diferença de tempo entre cada iteração.



Fonte: Produção do Autor.

Nota-se que a diferença de 0.7s é confirmada para todos os dados exceto para o primeiro e o segundo. A diferença entre o primeiro e o segundo dado é de 0.697s. Isto ocorre provavelmente devido às rotinas da inicialização do sistema. Essa diferença não é significativa em relação à média. Isto é denotado pelo baixo desvio padrão, conforme o valor apresentado na Tabela 5.14. Portanto, a primeira diferença pode ser considerada um *outlier,* i.e., ponto fora da curva.

Tabela 5.14 - Média, Desvio Padrão e Moda.

	Média	**Desvio Padrão**	**Moda**
Diferença entre cada iteração Tempo de envio	0.7	2.4577×10^{-4}	0.7

Fonte: Produção do Autor.

O resultado apresentado na Tabela 5.14 indica o alto nível de determinismo do sistema do módulo principal.

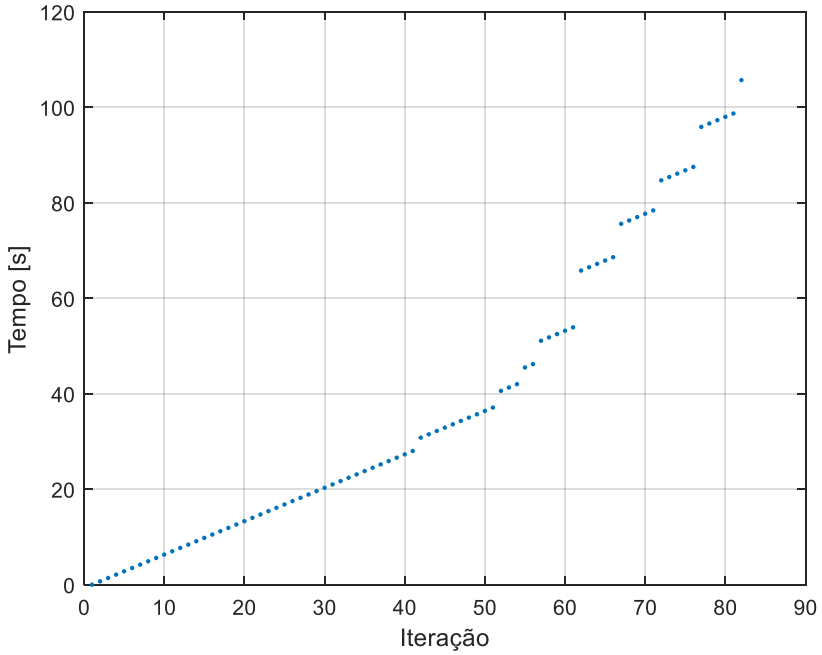
5.5.3. Análise temporal dos dados recebidos

A análise temporal das mensagens recebidas pelo RDC é realizada por meio da diferença de tempo de recebimento entre cada iteração. É analisado o comportamento do sistema ao receber os dados pela rede.

O conteúdo das mensagens recebidas é o tempo em que as mesmas são enviadas. Isso possibilita a análise do tempo de envio dos dados recebidos quando há falha na comunicação e perda de dados.

Nesta implementação, não há o recebimento de todos os dados enviados pelo módulo principal. São recebidos 82 dados de 150 enviados. Então, na Figura 5.40, é possível observar a sequência de dados, convertidos para segundos, no qual o recebimento é bem-sucedido.

Figura 5.40 - Tempo de envio dos dados recebidos.



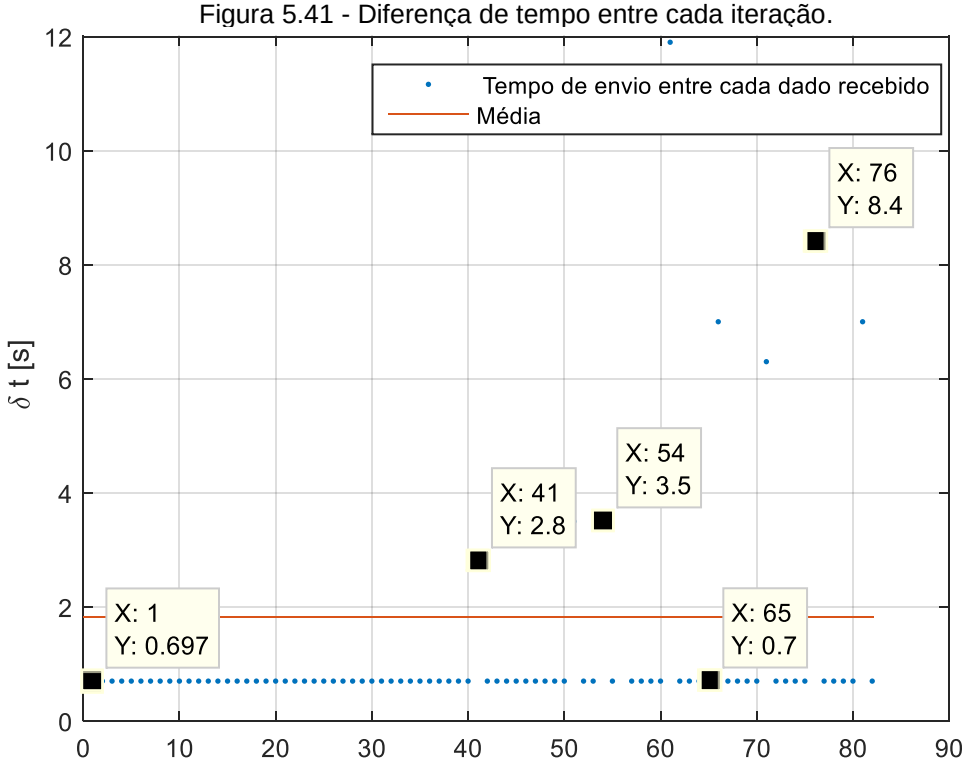
Fonte: Produção do Autor.

É provável que a perda de alguns dos dados enviados neste caso (caso 2b) aconteça devido à frequência que o RDC nota a existência de dados no canal/porta de comunicação e efetua o recebimento do mesmo. Como essa frequência é menor que a frequência da implementação anterior (caso A), isso sugere que ele não consiga notar a existência de dados no canal/porta de comunicação e efetuar o recebimento dos dados em tempo hábil, ou seja, retirar as mensagens/os dados do canal/porta, que faz com que o mesmo não tenha espaço disponível para receber novas mensagens. Como não há espaço no canal/porta, as mensagens enviadas pela partição “pidsca” para a IOP1 são perdidas quando a mesma tenta enviar para a rede. O reenvio dos dados perdidos não é solicitado pela rede, pois o protocolo de comunicação utilizado é o UDP.

Quando há espaço novamente no canal/porta, as mensagens enviadas no mesmo, se referem a uma outra chamada da função do sistema operacional utilizada. Como o conteúdo das mensagens enviadas é o tempo em que as mesmas são enviadas, é possível observar uma diferença de tempo de envio

maior entre os dados recebidos. Porém, essa diferença é sempre múltipla de 0.7s, que corresponde ao agendamento do módulo principal, responsável pelo envio dos dados, como mostra a Figura 5.41.

Figura 5.41 - Diferença de tempo entre cada iteração.

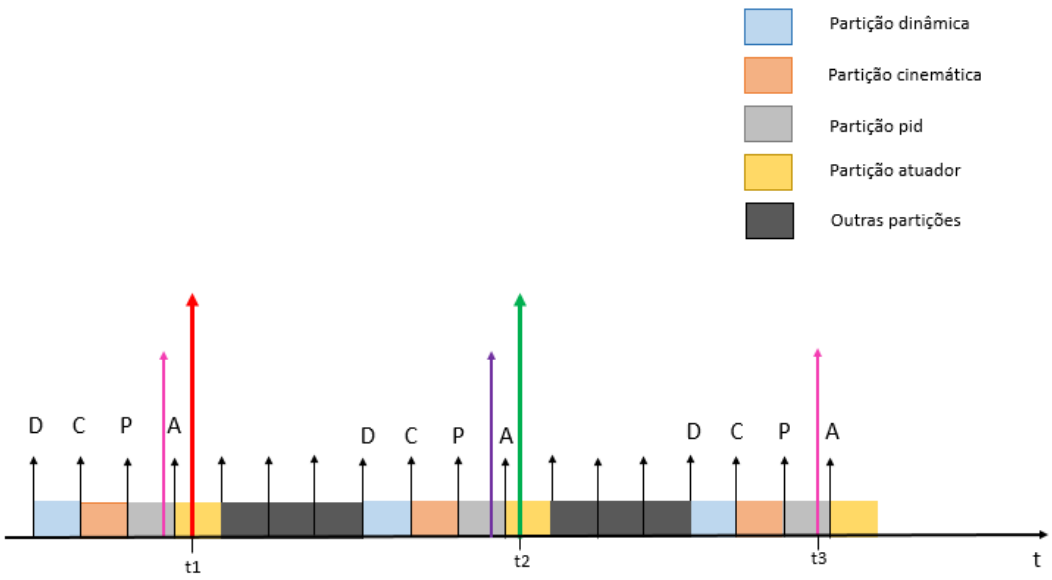


Fonte: Produção do Autor.

Como analisado na seção anterior, a diferença de tempo de envio entre a primeira e a segunda iteração, 0.697s, pode ser considerada um ponto fora da curva.

A perda de dados acontece provavelmente devido à falta de espaço no canal/porta, ocasionada pela frequência que o RDC efetua o recebimento dos dados. A Figura 5.42, a seguir, apresenta uma escala de tempo para observar a sequência de acontecimentos que provavelmente leva à ocorrência desse fenômeno. Para facilitar a observação, assume-se que o dado é enviado diretamente para a rede pela partição “pidsca”, desconsiderando a existência da partição IOP1.

Figura 5.42 - Escala de tempo.



Fonte: Produção do Autor.

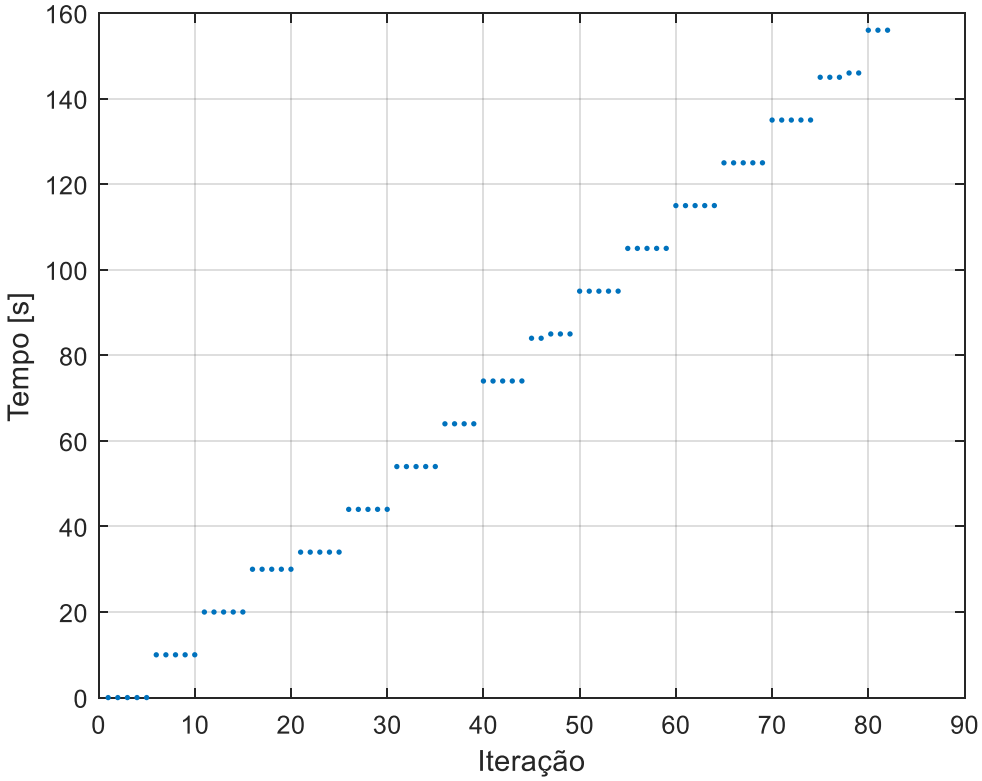
Após a mensagem ser enviada pela partição *"pidsca"*, primeira *flag* rosa, o canal/porta passa a não ter mais espaço, *flag* vermelha no instante t1. Quando a partição *"pidsca"* envia uma nova mensagem, a mesma é perdida, *flag* roxa, pois o canal/porta ainda está cheio. No instante t2, é liberado espaço no canal, *flag* verde. No instante t3, quando a partição *"pidsca"* envia uma nova mensagem, a mesma é enviada para a rede, *flag* rosa, pois há espaço no canal/porta de comunicação.

Ao fazer a análise do tempo em que cada dado é recebido, é preciso ter o tempo inicial definido, pois no RDC, o tempo de recebimento do primeiro dado é variável. Isso acontece devido à utilização da função do sistema operacional, que retorna o número de pulsos do relógio decorridos desde que o sistema é inicializado.

Considera-se então o tempo inicial de recebimento como zero segundos; sendo assim, é necessário fazer a correção do tempo de recebimento de todos os dados. Então desconta-se o tempo de recebimento do primeiro dado do tempo

de recebimento de todos os dados. A Figura 5.43, a seguir, apresenta o tempo, em segundos, em que cada dado é recebido após essa correção.

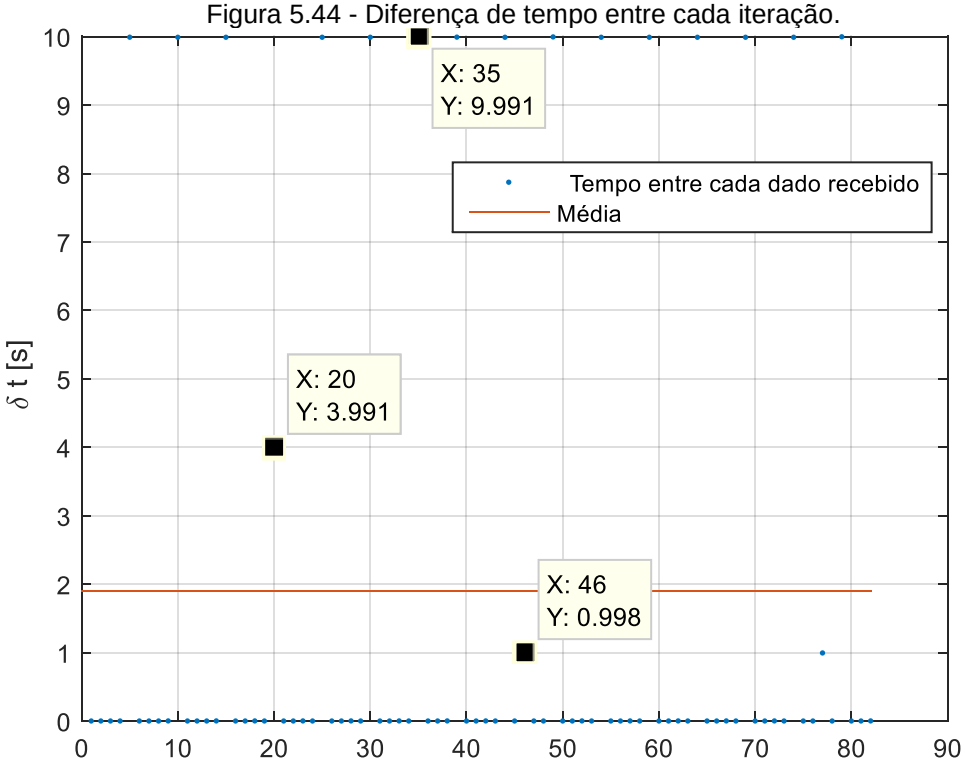
Figura 5.43 - Tempo de recebimento.



Fonte: Produção do Autor.

Na Figura 5.43, é possível observar que existem duas faixas de tempo para o recebimento dos dados, assim como nas implementações anteriores, uma faixa que representa a maior diferença de tempo de recebimento, (Δt 1), e uma faixa que representa a menor diferença de tempo de recebimento, (Δt 2), entre cada dado recebido. A Figura 5.44, a seguir, apresenta a diferença de tempo entre cada iteração.

Figura 5.44 - Diferença de tempo entre cada iteração.



Fonte: Produção do Autor.

A cada iteração com período de 0,7s, a partição “pidsca” do módulo principal envia um dado; já o RDC pode receber *n* dados a cada iteração de 0,1s de duração e período de 0.2s. Nota-se, que os valores da diferença de tempo de recebimento entre um dado e outro são bem próximos à valores múltiplos do período do RDC. Essas flutuações ocorrem provavelmente devido às incertezas temporais do hardware.

O RDC leva um tempo desconhecido e variável para perceber a disponibilidade dos dados na rede para recebimento. Quando percebe, já existem dados acumulados. Então, é provável que, enquanto há dados disponíveis para recebimento e perceptíveis ao RDC, o RDC os recebe, o que gera uma diferença quase nula entre o tempo de recebimento da maioria dos dados.

A Tabela 5.15, apresenta o desvio padrão e a moda da diferença de tempo de recebimento entre cada dado. O desvio padrão relativamente alto, é esperado devido à faixa de maior diferença do tempo de recebimento. E a moda é compatível com a existência da faixa de menor diferença do tempo de recebimento, que é mais representativa.

Tabela 5.15 - Desvio Padrão e Moda.

	Desvio Padrão	**Moda**
Diferença entre cada iteração Tempo de recebimento	3.8789	0.002

Fonte: Produção do Autor.

A Tabela 5.16 a seguir, apresenta a média, o desvio padrão, a moda, o valor mínimo e o valor máximo da faixa de maior diferença do tempo de recebimento dos dados, (Δt 1), assim como a quantidade de ocorrências da mesma.

Tabela 5.16 - Média, Desvio Padrão, Moda, Quantidade, Valor Mínimo e Máximo.

	Média	**Desvio Padrão**	**Moda**
Diferença entre cada iteração - Faixa de maior diferença do tempo de recebimento Δt 1	8.6592	3.1235	9.9910
	Quantidade	**Valor Mínimo**	**Valor Máximo**
	18	0.9960	9.9980

Fonte: Produção do Autor.

A Tabela 5.17 a seguir, apresenta a média, o desvio padrão, a moda, o valor mínimo e o valor máximo da faixa de menor diferença do tempo de recebimento dos dados, (Δt 2), assim como a quantidade de ocorrências da mesma.

Tabela 5.17 - Média, Desvio Padrão, Moda, Quantidade, Valor Mínimo e Máximo.

	Média	**Desvio Padrão**	**Moda**
Diferença entre cada iteração - Faixa de menor diferença do tempo de recebimento Δt 2	0.0022	0.000408	0.002
	Quantidade	**Valor Mínimo**	**Valor Máximo**
	63	0.002	0.003

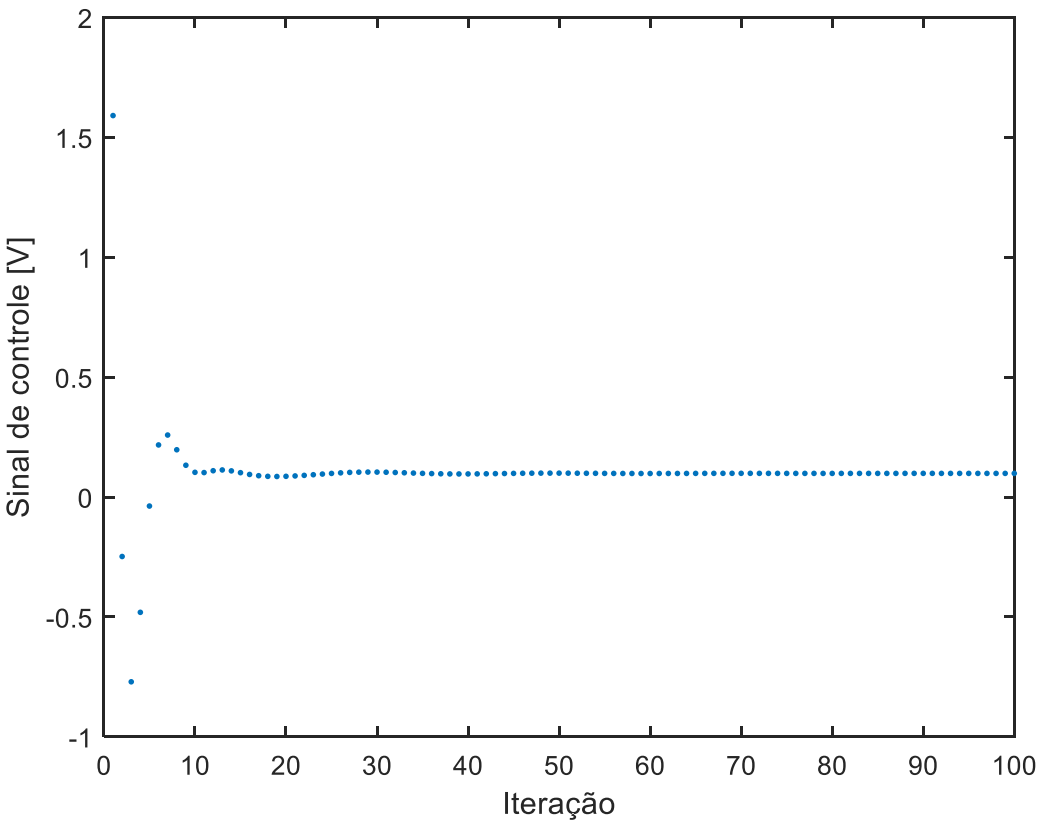
Fonte: Produção do Autor.

5.6. Implementação 3 - Caso de teste A

Nesta implementação, o módulo principal em modo falha envia os dados, sinal de controle do sistema MMA, para o RDC. O objetivo desta implementação é comprovar que o erro de computação (apresentado no Capítulo 3) inserido no sistema SCA e a reconfiguração do mesmo, ao tratar a condição de falha inserida, não interferem no funcionamento e na comunicação do sistema MMA com o RDC.

Essa implementação, tem um total de 100 dados enviados, na qual as mensagens enviadas são organizadas em fila e lidas em ordem (FIFO). Observa-se na Figura 5.45, a seguir, o sinal de controle enviado.

Figura 5.45 - Sinal de controle enviado.



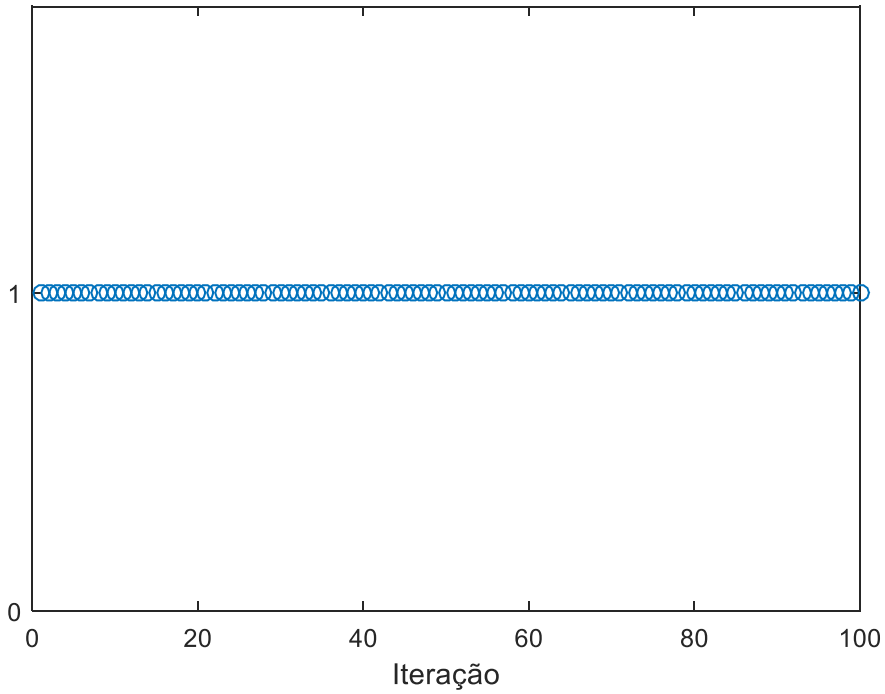
Fonte: Produção do Autor.

5.6.1. Verificação lógica dos dados

A verificação lógica dos dados é realizada por meio da comparação entre os dados enviados pelo módulo principal e recebidos pelo RDC. É verificado se os dados são entregues.

Ao se fazer a verificação dos dados recebidos, percebe-se que todos os dados são recebidos, como mostra a Figura 5.46, na qual o número 1 representa dados recebidos, e o número 0 representa dados perdidos.

Figura 5.46 - Análise dos dados enviados.

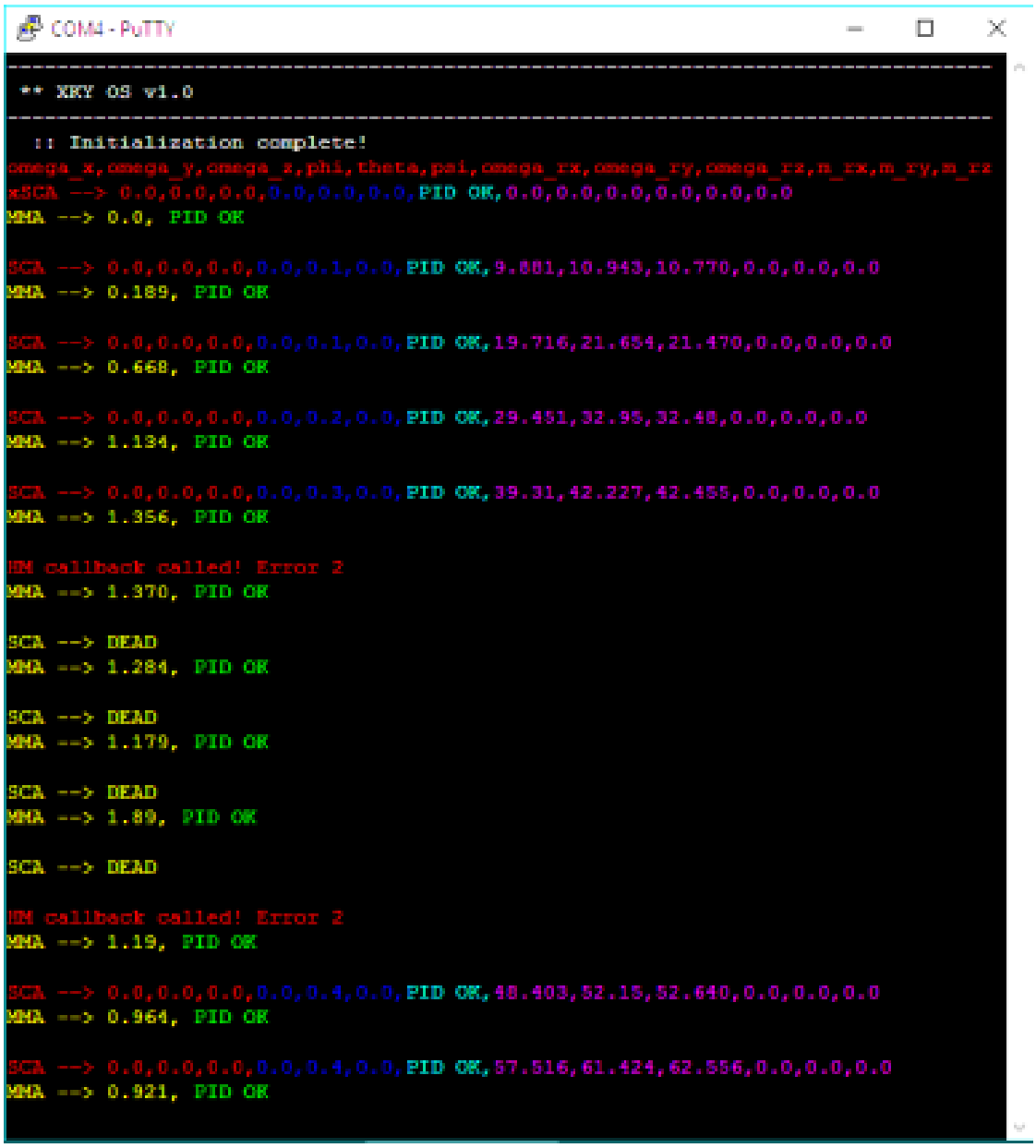


Fonte: Produção do Autor.

A Figura 5.47 apresenta os dados e as informações obtidas do módulo principal por (MORAES, 2017). Neste modo de operação, após 5 ciclos de execução, o erro de computação é inserido na partição (em vermelho) do sistema SCA, que entra em modo falha (*DEAD*), porém as partições do sistema MMA (em amarelo e verde) continuam operando normalmente.

Com o resultado da verificação lógica dos dados pode-se comprovar que o erro de computação inserido no sistema SCA e a sua reconfiguração ao tratar a condição de falha inserida, não interferem no funcionamento, Figura 5.47, e na comunicação do sistema MMA com o RDC, Figura 5.46.

Figura 5.47 - Kit em modo falhado.



Fonte: Moraes (2017).

5.7. Comparação dos resultados das implementações

A Tabela 5.18 apresenta a média, o desvio padrão e a moda da diferença de tempo de envio dos dados entre cada iteração das implementações realizadas. O baixo desvio padrão nas implementações indica o alto nível de determinismo do sistema do módulo principal. Sabe-se que, em todas as implementações a

configuração do módulo principal é a mesma. Nota-se que a partição “pidsca” do módulo principal apresentou comportamento idêntico no envio dos dados para a partição IOP1 responsável pelo envio de dados para a rede.

Tabela 5.18 - Diferença entre cada iteração - Tempo de envio.

Implementação	Média	Desvio Padrão	Moda
1.b	0.7	2.4577×10^{-4}	0.7
2.a	0.7	2.4577×10^{-4}	0.7
2.b	0.7	2.4577×10^{-4}	0.7
2.c	0.7	2.4577×10^{-4}	0.7

Fonte: Produção do Autor.

A Tabela 5.19 apresenta o desvio padrão, a moda e a porcentagem de dados recebidos da diferença de tempo de recebimento entre cada iteração das implementações realizadas.

É importante lembrar que em todas as implementações da Tabela 5.19, o módulo principal envia a mesma quantidade de dados e as análises são realizadas sem considerar o atraso desconhecido para o RDC receber o primeiro dado.

Na implementação 1, a plataforma utilizada para o RDC é o Arduino Uno; nas demais implementações apresentadas na Tabela 5.19, a plataforma de desenvolvimento utilizada para o RDC é o *Beagle Bone Black* BBB2*,* que permite definir o tempo de execução da partição que recebe os dados e variar o período da mesma.

Tabela 5.19 - Diferença entre cada iteração - Tempo de recebimento.

Implementação	Período do RDC	Desvio Padrão	Moda	Dados Recebidos
1.b	-	1.5868	0.005	100%
2.a	0.2 segundos	1.4594	0.002	100%
2.b	0.7 segundos	1.9180	0.002	84.6667%
2.c	1.0 segundos	3.8789	0.002	54.6667%

Fonte: Produção do Autor.

A moda de cada uma das implementações é compatível com a existência da faixa de menor diferença do tempo de recebimento. Nota-se que esse valor é maior na implementação que utiliza o Arduino.

Ao observar os casos de teste da implementação 2, nota-se que, quanto maior é o período do RDC, menor é a frequência de recebimento; logo, como esperado, um número maior de dados é perdido. Então, um RDC com frequência de leitura menor ou idêntica à frequência de envio, faz com que haja perda de dados durante a comunicação, pois não há o reenvio dos dados perdidos devido ao protocolo de comunicação utilizado, i.e., UDP.

Na Tabela 5.20 e na Tabela 5.21 são apresentadas a média, o desvio padrão, a moda, o valor mínimo e o valor máximo das faixas que representam a maior e a menor diferença de tempo de recebimento dos dados, respectivamente.

Ao observar os dados na Tabela 5.20, i.e., a média, o desvio padrão, a moda, o valor mínimo e o valor máximo, da implementação 1-caso A e da implementação 2-caso B, nota-se que o comportamento do RDC implementado no Arduino Uno é semelhante ao do RDC implementado na plataforma *Beagle Bone Black*, configurado com um período de 0.7s para recebimento dos dados; sabe-se que o período/frequência de recebimento dos dados não está configurado e definido no RDC implementado no Arduino Uno. Então, provavelmente, a existência da

faixa que representa a maior diferença temporal de recebimento dos dados é definida pela partição IOP1 do módulo principal, já que a partição “pidsca” do módulo principal envia os dados para a IOP1 de forma contínua a cada iteração.

Tabela 5.20 - Faixa que representa a maior diferença de tempo de recebimento, (Δt 1).

Implementação	Período do RDC	Média	Desvio Padrão	Moda	Valor Mínimo	Valor Máximo
1.b	-	3.7579	1.8129	4.8830	0.6060	4.8980
2.a	0.2	2.0122	2.0067	0.3910	0.1930	4.9980
2.b	0.7	4.1230	1.5728	4.8910	0.6930	4.8980
2.c	1.0	8.6592	3.1235	9.9910	0.9960	9.9980

Fonte: Produção do Autor.

Tabela 5.21 - Faixa que representa a menor diferença de tempo de recebimento, (Δt 2).

Implementação	Período do RDC	Média	Desvio Padrão	Moda	Valor Mínimo	Valor Máximo
1.b	-	0.0046	0.0018	0.005	0.001	0.0120
2.a	0.2	0.0022	0.000406	0.002	0.002	0.003
2.b	0.7	0.0022	0.000415	0.002	0.002	0.003
2.c	1.0	0.0022	0.000408	0.002	0.002	0.003

Fonte: Produção do Autor.

Porém, ao observar os resultados dessas implementações (1.b e 2.b) na Tabela 5.19, percebe-se que na implementação 2-caso B há uma perda de dados. É provável que essa perda de dados aconteça devido à frequência com que o RDC nota a existência de dados no canal/porta de comunicação e efetua o recebimento do mesmo, ou seja o período de recebimento dos dados definido no agendamento do RDC. Pois, mesmo havendo dados disponíveis para leitura

no canal/porta, o RDC implementado no *Beagle Bone Black* tem que obedecer ao seu agendamento, o que atrasa a leitura dos dados e consequentemente leva ao fenômeno do canal/porta cheio; enquanto que, no RDC implementado no Arduino Uno, não há agendamento para definir quando o mesmo pode realizar a leitura dos dados. O cumprimento do agendamento pode ser notado ao observar os valores mínimos e máximos da Tabela 5.20 dos casos de testes da implementação 2, que são valores próximos aos múltiplos do período de recebimento dos dados definido no agendamento do RDC. Essas flutuações, i.e., valores próximos aos múltiplos do período de recebimento, ocorrem provavelmente devido às incertezas temporais do *hardware*.

Logo, além de ser definida, provavelmente, pela partição IOP1 do módulo principal, a faixa que representa a maior diferença de recebimento dos dados também é influenciada pelo período de recebimento dos dados definido no agendamento do RDC, assim como, provavelmente, também pela IOP2 do RDC.

Após observar a possível influência das IOP's, pode-se considerar que o tempo, desconhecido e variável para o RDC perceber a disponibilidade dos dados na rede para recebimento, também sofre influência das mesmas. Pois, no RDC, independente da plataforma de desenvolvimento utilizada, o tempo de recebimento dos dados não é linearmente crescente como o envio dos mesmos pela partição "pidsca" do módulo principal.

Ao comparar as implementações (1.a e 3.a) nas quais a plataforma utilizada para o RDC é o Arduino Uno, e o dado enviado é o sinal de controle; é importante lembrar que: na implementação 1-caso A, envia-se a cada iteração 3 dados: sinais de controle dos eixos X, Y e Z, respectivamente; e na implementação 3-caso A, envia-se apenas um sinal de controle. O envio de apenas um dado por iteração, gera um menor fluxo de dados no canal/porta da implementação 3-caso A; então, neste caso, o RDC, mesmo ao levar tempo desconhecido e variável para receber o primeiro dado, é eficiente no recebimento de todos os dados. Assim, o provável fenômeno de canal/porta cheio não acontece e, consequentemente, não existe a perda de dados.

6 DIFICULDADES ENCONTRADAS

6.1. Dificuldades gerais

A fim de realizar as implementações para alcançar o objetivo proposto para este trabalho, inúmeras dificuldades foram encontradas:

1) Inicialmente, foi preciso entender todo o sistema desenvolvido em (MORAES, 2017). O sistema desenvolvido e embarcado no demonstrador DIMA é um sistema complexo. Ele é dividido e separado em partições; então, é necessário entender como as partições se comunicam, e se atentar aos detalhes de configuração das portas e canais criados entre elas, assim como as limitações dos mesmos.

2) A documentação disponível sobre o sistema operacional é rara e limitada. Após a participação em um breve curso, proporcionado pelo Eng. Sérgio Penna na Embraer, realizado pela GMV sobre uma versão mais atualizada (2) do sistema, um novo documento sobre o mesmo foi obtido. Porém, a versão utilizada neste trabalho foi a primeira (1). A nova versão possui modificações significativas e difere em configurações, chamadas de funções do sistema, entre outros. Além disso, o grupo de desenvolvedores da primeira versão não estava mais disponível para contato, o que dificultou muito a troca de informações para as sanar dúvidas referentes à erros obtidos.

3) Os erros obtidos nas tentativas de realização das implementações indicavam apenas valores numéricos, por exemplo: erro 1, erro 2. Saber o que cada erro significa é impraticável sem a referenciação dos mesmos na documentação disponível, o que dificultou bastante o andamento das implementações. Na maior parte do tempo de desenvolvimento do trabalho o protocolo utilizado para o envio dos pacotes na rede era desconhecido. O conhecimento do mesmo é importante para a comunicação com o Arduino.

4) Um contato virtual, possibilitado através do Eng. Sergio Penna com o Eng. Miguel Barros, engenheiro de software da GMV e integrante da equipe de

desenvolvimento da versão atual do sistema operacional, foi essencial para a conclusão das implementações deste trabalho. Nesse contato, a maioria das dificuldades e dúvidas foram sanadas. Sem essa troca de informações e ajuda imensa do Eng. Miguel Barros, muito provavelmente seria impossível a continuação e a conclusão deste trabalho.

6.2. Dificuldades pontuais

Dentro de todas as dificuldades gerais apresentadas acima, existiram diversas dificuldades pontuais durante o desenvolvimento do trabalho. Estão listadas a seguir algumas das dificuldades com um breve parecer dos acontecimentos.

6.2.1. Atualização do *Windows*

O embarque do sistema no demonstrador às vezes era impedido pelo sistema operacional do notebook, Windows, quando o mesmo atualizava e modificava algumas de suas configurações, por exemplo *firewall,* que impedia o funcionamento do aplicativo servidor Tftpd32. A atualização do sistema operacional também prejudicava o funcionamento da máquina virtual utilizada para a execução do ambiente AIR. O bloqueio da atualização automática do Windows nem sempre era suficiente para impedir os problemas causados pelo mesmo.

6.2.2. Configuração das partições, portas e canais de comunicação

A partição responsável pelo gerenciamento da comunicação, partição IOP, precisa ser configurada com atenção redobrada. Qualquer erro na configuração gera o insucesso na comunicação e nem sempre os erros são evidenciados.

Na configuração, arquivo xml – Figura 6.1, do dispositivo que envia os dados: a porta de envio de dados (“pid2iop”) da partição emissora (“pidsca” ou “pidmma”) para a partição IOP1 tem que estar associada com o dispositivo que irá receber dados, chamado no arquivo de configuração como dispositivo lógico.

Figura 6.1 - Parte da configuração XML da partição IOP1 do módulo principal.

```
<LogicalDevices>     <!-- nome do dispositivo que irá receber os dados -->
    <Device Id="2" Name="RDC" />
</LogicalDevices>

<RemotePorts>
    <Port Name="pid2iop" LogicalDeviceId="2"/>
</RemotePorts>
```

Fonte: Produção do Autor.

Quando o dispositivo que recebe os dados é o *Beagle Bone Black* BBB2, a configuração, arquivo xml – Figura 6.2, da partição IOP2 é diferente: a porta (“iop2rec”) da partição (“*receiver*”) que recebe os dados da IOP2 não pode estar associada com o dispositivo que irá enviar os dados, chamado no arquivo de configuração como dispositivo lógico. Se a configuração não estiver conforme descrita acima, a comunicação não é realizada.

Figura 6.2 - Parte da configuração XML da partição IOP2 do RDC.

```
<LogicalDevices>
    <Device Id="1" Name="BBB1" /> <!-- nome do dispositivo que irá enviar os dados -->
</LogicalDevices>

<RemotePorts>
    <Port Name="iop2rec" />
</RemotePorts>
```

Fonte: Produção do Autor.

A nomeação dos canais e das partições, nos códigos, não pode ser feita com a utilização de letras maiúsculas.

O tamanho do canal e o número máximo de mensagens enviadas por ele tem que ser configurados de acordo com o fluxo de dados da aplicação, o que pode parecer óbvio, mas não é. Pois, mesmo com a comunicação configurada corretamente e com a existência de espaço para o envio de alguma mensagem, nenhuma mensagem é enviada se a configuração do tamanho do canal estiver incorreta. Por exemplo: ao tentar enviar três mensagens por vez no canal, nenhuma mensagem é enviada, entretanto, há o sucesso no envio da mensagem quando se tenta enviar apenas uma por vez. Isso pode camuflar o problema e o

mesmo ser interpretado como outro tipo de erro; então, adequar o tamanho do canal de acordo com a aplicação é essencial.

6.2.3. Interpretação dos dados no Arduino

É importante se atentar ao tipo e ao tamanho dos dados recebidos, i.e., *float*, *char, int*, para se ter a leitura correta dos dados no Arduino.

Os dados enviados pelo módulo principal são do tipo *float*. Para o Arduino as mensagens recebidas são um vetor de caracteres, i.e., um vetor do tipo *char*. Então é necessário realizar no Arduino um processo de conversão de dados para se ter a leitura correta dos dados recebidos.

A solução encontrada foi a utilização da função *memcpy ()*, cuja a finalidade é copiar blocos de memória. Com os seus 3 parâmetros: destino, origem e número de bytes a serem copiados, foi possível copiar o conteúdo da mensagem recebida para um destino em que a interpretação correta dessa mensagem é realizada. Assim, o dado recebido foi interpretado como realmente ele foi enviado, do tipo *float.*

6.2.4. Observação dos dados

A observação dos dados enviados pelo módulo principal, através do terminal serial, por motivo desconhecido, se tornou enganosa. Mesmo ao utilizar terminais seriais diferentes para observar as mensagens enviadas, os valores negativos eram substituídos por zero e as casas decimais eram trocadas sem padrão definido. Esses erros só puderam ser descobertos através da verificação lógica dos dados e com a observação dos dados recebidos pelo RDC implementado no Arduino.

A princípio, acreditava-se que havia uma falha na comunicação com uma perda enorme de dados, o que levou à observação minuciosa de cada mensagem enviada e recebida. Percebeu-se que alguns dos dados recebidos nunca foram enviados. Com isso em mente, e ao observar as mensagens novamente, obteve-se a primeira conclusão: no terminal serial aparecia o valor zero para os dados enviados que verdadeiramente tinham valores negativos. É importante salientar

que os dados eram enviados corretamente pela rede e dentro das partições do demonstrador, porém a visualização dos valores dos mesmos era enganosa.

Após a correção, dados recebidos que nunca foram enviados continuaram existindo. Ao observar novamente, obteve-se a segunda conclusão: as casas decimais eram trocadas sem padrão definido. Por exemplo, ao enviar 25.2678, aparecia como enviado 25.6278, ou 25.2687, etc.

A solução dos problemas foi a transformação dos valores de forma matemática, para que os mesmos ao serem apresentados no terminal serial correspondessem aos valores reais. É importante salientar que os dados enviados continuaram da mesma forma; a transformação foi apenas na visualização. Desse modo, o tamanho das mensagens enviadas não é alterado.

As dificuldades encontradas durante a realização deste trabalho mostram que todo o processo de comunicação e de desenvolvimento das implementações, assim como a obtenção dos resultados das mesmas, não é nada trivial.

7 COMENTÁRIOS FINAIS, CONCLUSÕES E SUGESTÕES PARA TRABALHOS FUTUROS

7.1. Comentários finais e conclusões

O trabalho apresentado propôs realizar uma apreciação do RDC em uma arquitetura DIMA aplicado ao controle de atitude da PMM em modo normal. Isto incluiu, acrescentar e realizar a comunicação em rede do módulo principal com o concentrador de dados remoto no Demonstrador DIMA. Isto também incluiu realizar a comparação dos resultados diretamente do módulo principal (modo normal e modo falha) com os resultados via RDC após a comunicação. Ambos foram feitos com sucesso.

As dificuldades encontradas durante a realização deste trabalho mostram que todo o processo de comunicação e de desenvolvimento das implementações, assim como a obtenção dos resultados das mesmas, não é trivial. O desenvolvimento em arquitetura DIMA é complexo.

A análise das alterações dos dados enviados pela rede Ethernet, que ocorrem durante a comunicação do módulo principal com o RDC é de grande importância, pois tais efeitos podem impedir o sistema de atingir o desempenho desejado. Neste trabalho, a comunicação foi realizada em malha aberta e o fluxo de dados dentro das partições do módulo principal continuou inalterado, sem prejudicar o sistema. Foi possível observar os efeitos da comunicação nos dados enviados e foi realizada uma simulação de um dos efeitos (perda de dados), para comprovar que o efeito degrada o desempenho ($\%M_p$) do sistema quando se trata de malha fechada.

Não há reenvio dos dados perdidos durante a comunicação, pois o protocolo utilizado é o UDP e, com o mesmo, normalmente, não existe uma solicitação de reenvio de pacote de dados.

O baixo desvio padrão do tempo de envio dos dados da partição “pidsca”, das implementações realizadas, indica alto nível de determinismo do sistema do módulo principal.

As partições IOP’s, provavelmente, influenciam o tempo de recebimento dos dados, e também definem a existência das faixas que representam as diferenças temporais de recebimento dos dados. Isso prejudica o comportamento correto de um sistema de tempo real pois, além de depender da integridade dos resultados obtidos, o mesmo também depende dos valores de tempo em que os resultados são produzidos.

O RDC, mesmo ao levar tempo desconhecido e variável para receber o primeiro dado, provavelmente pela influência da(s) IOP(s), consegue receber todos os dados quando o mesmo é mais ágil que o módulo principal e, quando o número de dados enviados por vez é menor.

O acréscimo da partição IOP no sistema do módulo principal e a realização das alterações necessárias para implementar a comunicação do módulo principal com o RDC prejudicam o desempenho do sistema em malha fechada dentro do módulo principal. Isso é percebido ao comparar com a resposta apresentada em (MORAES, 2017), sem as modificações necessárias para realizar a comunicação.

Apesar do *dead reckoning* ser bastante utilizado e conhecido como uma boa solução, no particular intervalo de tempo do sistema apresentado neste trabalho isso não acontece. O desempenho ($\%M_p$) do sistema é pior ao utilizar o *dead reckoning* do que ao utilizar o *dead zero*.

No caso do demonstrador deste trabalho, que se trata de um demonstrador com objetivo didático, a rede utilizada para realizar comunicação é uma rede não determinística. Em um sistema aviônico distribuído, a rede tem bastante impacto no mesmo, principalmente ao se tratar de um sistema reconfigurável. Todo o sistema e sua reconfigurabilidade depende das capacidades da rede. Os

impactos da rede na reconfiguração do sistema, em uma arquitetura DIMA, precisam e ainda estão sendo considerados e estudados.

As explicações apresentadas neste trabalho, devem e podem ser melhor verificadas através de ferramentas específicas e mais sofisticadas. Apesar das explicações serem as melhores a oferecer, cabem análises mais profundas dos fenômenos percebidos. Estas são detalhadas na próxima seção.

7.2. Sugestões para trabalhos futuros

Sugere-se para a continuação deste trabalho no demonstrador DIMA:

a) O estudo e a verificação, mais aprofundada, da existência de um tamanho de canal de comunicação ideal para cada uma das implementações, apresentadas no Capítulo 4, com o objetivo de evitar ou reduzir a perda de dados, assim como o acréscimo de uma lógica otimizada de verificação dos dados e reenvio dos dados perdidos na comunicação. E a análise do efeito do acréscimo dessa lógica no desempenho do módulo principal e de todo sistema.
b) Ou o estudo e a verificação de um RDC ideal, diante as limitações existentes, i.e., canal, rede, que seja capaz de ler a maior quantidade de dados possíveis, idealmente todos os dados, para que não haja perda de dados devido ao provável fenômeno apresentado nas implementações deste trabalho.
c) Efetuar a sincronização de relógios para a comunicação em malha fechada, com as outras formas de realização do RDC, e realizar estratégias de controle capaz de lidar com o atraso causado pela comunicação em rede.
d) Separar o SCA do restante do sistema. Implementar em plataformas de desenvolvimento diferentes: o controle, o(s) sensor(es), atuador(es) e planta, e o RDC. Dessa forma é possível avaliar no demonstrador a arquitetura DIMA de uma forma mais próxima à realidade.

REFERÊNCIAS BIBLIOGRÁFICAS

AERONAUTICAL RADIO, INCORPORATED (ARINC). **ARINC specification 650-1**: design guidance for integrated modular avionics. Maryland: ARINC, 1997. 88 p.

AERONAUTICAL RADIO, INCORPORATED (ARINC). **ARINC specification 650**: integrated modular avionics packaging and interfaces. Maryland:ARINC, 1994. 119 p.

AERONAUTICAL RADIO, INCORPORATED (ARINC). **ARINC specification 652**: guidance for avionics software management. Maryland: ARINC, 1993. 73 p.

AERONAUTICAL RADIO, INCORPORATED (ARINC) **ARINC specification 653 P0**: avionics application software standard interface, overview of Arinc 653. Maryland: ARINC, 2013. 35 p.

AERONAUTICAL RADIO, INCORPORATED (ARINC). **ARINC specification 653 P1-3**: avionics application software standard interface: equired services. Maryland: ARINC, 2010. 269 p.

AERONAUTICAL RADIO, INCORPORATED (ARINC). **ARINC specification 653 P2-2**: avionics application software standard interface part 2, extended services. Maryland: ARINC, 2007. 211 p.

AERONAUTICAL RADIO, INCORPORATED (ARINC). **ARINC specification 653 P3A**: avionics application software standard interface, part 3a: conformity test secification for arinc 653 required services. Maryland: ARINC, 2006. 411 p.

AERONAUTICAL RADIO, INCORPORATED (ARINC). **ARINC specification 653 P4**: avionics application software standard interface Part 4, subset services. Maryland: ARINC, 2012. 85 p.

AERONAUTICAL RADIO, INCORPORATED (ARINC). **ARINC specification 653 P5**: avionics application software standard interface Part 5, core software recommended capabilities. Maryland: ARINC, 2014. 21 p.

AERONAUTICAL RADIO, INCORPORATED (ARINC). **ARINC specification 654**: Environmental Design Guidelines for Integrated Modular Avionics Packaging and Interfaces. Maryland, December 1994. 51 p.

AERONAUTICAL RADIO, INCORPORATED (ARINC). **ARINC specification 664P1-1**: aircraft data network part 1, systems concepts and overview. Maryland: ARINC, 2006. 51 p.

AERONAUTICAL RADIO, INCORPORATED (ARINC). **ARINC specification 664P2-1**: aircraft data network Part 2, ethernet physical and data link layer specification. Maryland: ARINC, 2006. 98 p.

AERONAUTICAL RADIO, INCORPORATED (ARINC). **ARINC specification 664P3-1**: aircraft data network Part 3, internet-based protocols and services. Maryland: ARINC, 2004. 90 p.

AERONAUTICAL RADIO, INCORPORATED (ARINC). **ARINC specification 664P4-2**: aircraft data network Part 4, internet-based address structure & assigned numbers. Maryland: ARINC, 2007. 60 p.

AERONAUTICAL RADIO, INCORPORATED (ARINC). **ARINC specification 664P5**: aircraft data network Part 5, network domain characteristics and interconnection. Maryland: ARINC, 2005. 125 p.

AERONAUTICAL RADIO, INCORPORATED (ARINC). **ARINC specification 664P7**: aircraft data network Part 7, Avionics Full Duplex Switched Ethernet (AFDX) network. Maryland: ARINC, 2005. 145 p.

AERONAUTICAL RADIO, INCORPORATED (ARINC). **ARINC specification 664P8**: aircraft data network Part 8, interoperation with non-IP protocols and services. Maryland: ARINC, 2005. 81 p.

ANNIGHÖFER, B. Model-driven development and simulation of Integrated Modular Avionics (IMA) architectures. **Simulation Notes Europe**, v.28, n.2, 2018.

BAGGERMAN, C.; MCCABE, M.; VERMA, D. **Avionics system architecture for NASA Orion vehicle**. Warrendale: SAE International, 2009.

BARROS, M. T. et al. **Distributed integrated modular avionics**. Lisboa: North Atlantic Treaty Organization Science and Technology Organization, 2018. Disponivel em: <https://www.sto.nato.int/publications/STO%20Meeting%20Proceedings/STO-MP-IST-166/MP-IST-166-03.pdf>.

CHEN, J.; DU, C.; HAN, P. Scheduling independent partitions in integrated modular avionics systems. **PLOS ONE**, 2016.

DEROCHE, E.; SCHARBARG, J.-L.; FRABOUL, C. A greedy heuristic for distributing hard real-time applications on an IMA architecture.In: IEEE INTERNATIONAL SYMPOSIUM ON INDUSTRIAL EMBEDDED SYSTEMS, 12., 2017, Toulouse. **Proceedings…** 2017.

DINIZ, N.; RUFINO, J. **ARINC 653 in space:** data systems in aerospace-conference. Edinburgh: ESA, 2005.

DOSS, M. et al. Migration of integrated modular avionics to space. In: DIGITAL AVIONICS SYSTEMS CONFERENCE, 15.,1996, Atlanta. **Proceedings…** IEEE/AIAA, 1996.

FARINES, J.; FRAGA, J. S.; OLIVEIRA, R. S. **Sistemas de tempo real**. Florianópolis: UFSC, 2000. 201 p.

FUCHSEN, R. Preparing the next generation of IMA- a new technology for the scarlett program. In: DIGITAL AVIONICS SYSTEMS CONFERENCE, 28., 2009. **Proceedings…** Orlando: IEEE/AIAA, 2009.

GASKA, T.; WATKIN, C.; CHEN, Y. Integrated modular avionics - past,present, and future. **IEEE A&E Systems Magazine**, p. 12-23, 2015.

GMV INNOVATING SOLUTIONS. **User guide XKY**. Lisboa: [s.n.], 2018. 113 p.

HOLANDA, J. Single modular on-board computer for space applications. In: SIMPÓSIO AEROESPACIAL BRASILEIRO, 2014, São José dos Campos. **Anais…** 2014.

802.3-2018 - IEEE Standard for Ethernet. **Institute of Electrical and Electronics Engineers**, New York, p. 5600, 2018.

JAKOLJEVIC, M.; ADEMAJ, A. Distributed IMA: use cases for embedded platforms. In: DIGITAL AVIONICS SYSTEMS CONFERENCE, 34., 2015. **Proceedings…** IEEE/AIAA, 2015.

JAKOVLJEVIC, M.; ADEMAJ, A. 2ND generation IMA-extended virtualization capabilities for optimized architectures. In: DIGITAL AVIONICS SYSTEMS CONFERENCE, 32., 2013. **Proceedings…** Syracuse: IEEE/AIAA. 2013.

KOPETZ, H. **Real-time systems:** design principles for distributed embedded applications. Boston: Kluwer Academic, 1997.

LIU, C. L.; LAYLAND, J. W. Scheduling algorithms for multiprogramming in a hard-real-time enviroment. **Journal of Association for Computing Machinery**, v. 20, n. 1, p. 46-61, 1973.

MARET, T. A. Avionics systems hosted on a distributed modular electronics Large scale demonstrator for multiple types of aircraft. In: AERODAYS, 2015, Londres, Inglaterra. **Proceedings…** 2015.

MCCABE, M.; BAGGERMAN, C.; VERMA, D. Avionics architecture interface considerations between constellation vehicles. In: DIGITAL AVIONICS SYSTEMS CONFERENCE, 28., 2009. **Proceedings…** Orlando: IEEE/AIAA, 2009.

MOIR, I.; SEABRIDGE, A. G. **Military avionics systems**. [S.l.]: John Wiley & Sons, 2006.

MORAES, A. L. O. **Apreciaçao, simulação e implementação da Arquitetura IMA Distribuída (DIMA) e sua aplicação a sistemas espaciais**. Dissertação (Engenharia e Gerenciamento de Sistemas Espaciais) - Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2017

OLIVEIRA JUNIOR, E. M. D. **Estudo dos algoritmos Welch-Lynch (FTM), Fault-Tolerant Average (FTA) e filtro de Kalman (FK) para sincronização de relógios e suas influências sobre um sistema de controle.** Dissertação (Mestrado em Engenharia e Tecnologia Espaciais/Mecânica Espacial e Controle) - Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2010.

POSTEL, J. **Transmission control protocol**. USC/Information Sciences Institute, 1980. Disponivel em: <https://tools.ietf.org/html/rfc761>.

POSTEL, J. **User datagram protocol**. USC/Information Sciences Institute, 1980. Disponivel em: <https://tools.ietf.org/html/rfc768>.

PRISAZNUK, P. J. Integrated modular avionics. In: NATIONAL AEROSPACE AND ELETRONICS CONFERENCE, 1992. **Proceedings…** Piscataway: IEEE/AIAA. 1992.

PRISAZNUK, P. J. ARINC 653 role in Integrated Modular Avionics (IMA). In: IEEE/AIAA DIGITAL AVIONICS SYSTEMS CONFERENCE, 27., 2008, Annapolis. **Proceedings…** IEEE, 2008.

RUFFINO, J. et al. AIR: technology innovation for future spacecraft onboard computing systems. In: **IEEE R8 Eurocon,** 2011, Lisboa. **Proceedings…** 2011.

RUFINO, J. AIR technology: a step towards ARINC 653 in space. In: DATA SYSTEMS IN AEROSPACE CONFERENCE, 2009, Istanbul. **Proceedings…** 2009.

SILBERSCHATZ, A.; GALVIN, P. B.; GAGNE, G. **Operating system concepts**. 7.ed. New York: John Wiley & Sons, 2005. 886 p.

SILVA, C. D. C. **Integrated modular avionics for space applications: i**nput/output module. 2012. 84p. Dissetarção (Mestrado em Engenharia Aeroespacial) - Universidade Técnica de Lisboa, Lisboa, 2012.

SPITZER, C. R.; FERRELL, U.; FERRELL, T. **Digital avionics handbook**. 3.ed. [S.l.]: CRC Press, 2015.

STANKOVIC, J. A. Misconceptions about real-time computing. **IEEE Computer**, v. 21, p. 10, 1988.

TAGAWA, G. B. S. **Estudo de um controlador de atitude em um simulador de Aviôniva Modular Integrada (IMA) ao Satélite Amazônia-1**. Dissertação (Mecânica Espacial e Controle) - Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2013. Disponivel em: <http://urlib.net/8JMKD3MGP7W/3DGNKP2>.

TODD, T.; NITSCHE, T. **Remote data concentrator**. US8996735B2, 31 Mar. 2015.

WANG, H.; NIU, W. A review on key technologies of the distributed integrated modular. **International Journal of Wireless Information Networks**, v.25, p.358-369, 2018.

WANG, Y. et al. Safety assessment process optimization for integrated modular avionics. **IEEE Aerospace and Electronic Systems Magazine**, v.34, n.11, p.58-67, 2019.

WATKINS, C. B.; WALTER, R. Transition from federated avionics architectures to integrated modular avionics. In: DIGITAL AVIONICS SYSTEMS CONFERENCE, 26., 2007, Dallas. **Proceedings…** IEEE/AIAA, 2007.

WILKINSON, C. IMA aircraft improvements. **IEEE A&E Systems Magazine**, v.20, n.9, p. 11-17, 2005.

WINDSOR, J.; DEREDEMPT, M.-H.; DE-FERLUC, R. Integrated modular avionics for spacecraft- user requirements, architecture and role definition. In: DIGITAL AVIONICS SYSTEMS CONFERENCE, 30., 2011, Seatle. **Proceedings…** IEEE/AIAA, 2011.

WINDSOR, J.; HJORTNAES, K. Time and space partitioning in spacecraft avionics. IEEE INTERNATIONAL CONFERENCE ON SPACE MISSION CHALLENGES FOR INFORMATION TECHNOLOGY, 2009. **Proceedings…** 2009.