Daniel Curcio Lott Guimarães

Metodologia para desenvolvimento de subsistemas de TT&C em CubeSats

Belo Horizonte

24 de fevereiro de 2022

Daniel Curcio Lott Guimarães

Metodologia para desenvolvimento de subsistemas de TT&C em CubeSats

Monografia apresentada junto ao Curso de Engenharia Aeroespacial Bacharelado da Universidade Federal de Minas Gerais - Departamento de Engenharia Mecância, como requisito parcial à obtenção do título de Engenheiro Aeroespacial

Universidade Federal de Minas Gerais

Escola de Engenharia

Departamento de Engenharia Mecânica

Orientador: Dra. Maria Cecília Pereira de Faria

Coorientador: Dr. Ricardo Luiz da Silva Adriano

Belo Horizonte

24 de fevereiro de 2022

Guimarães, Daniel C. L. Metodologia para desenvolvimento de subsistemas de TT&C em CubeSats/ Daniel Curcio Lott Guimarães. – Belo Horizonte, 24 de fevereiro de 2022- 77p. : il. (algumas color.) ; 30 cm.
Orientador: Dra. Maria Cecília Pereira de Faria
Monografia – Universidade Federal de Minas Gerais
Escola de Engenharia
Departamento de Engenharia Mecânica, 24 de fevereiro de 2022.
1. TT&C. 2. CubeSats. 3. Metodologia de Projeto. I. Dra. Maria Cecília Pereira de Faria. II. Universidade Federal de Minas Gerais. III. Escola de Engenharia. IV. Metodologia para desenvolvimento de subsistemas de TT&C em CubeSats.

ATA DE DEFESA PÚBLICA DO TRABALHO DE CONCLUSÃO DE CURSO

Aos 11 dias do mês de fevereiro de 2022, às 17 horas, em sessão pública virtual, na presença da Banca Examinadora presidida pela Professora Maria Cecilia Pereira (coorientadora, DEMEC/UFMG) e composta pelos examinadores:

1. Orientador: Ricardo Luiz da Silva Adriano (DEE/UFMG)
2. Membro 1 da Banca Examinadora: Diogo Batista de Oliveira (DEE/UFMG)
3. Membro 2 da Banca Examinadora: Dimas Abreu Archanjo Dutra

o aluno Daniel Curcio Lott Guimarães apresentou o Trabalho de Conclusão de Curso intitulado: **Metodologia para desenvolvimento de subsistemas de TT&C em CubeSats** como requisito curricular indispensável para a integralização do Curso de Bacharelado em Engenharia Aeroespacial. Após reunião em sessão reservada, a Banca Examinadora deliberou e decidiu pela **Aprovação** do referido trabalho, divulgando o resultado formalmente ao aluno e demais presentes e eu, na qualidade de Presidente da Banca, lavrei a presente ata que será assinada por mim, pelos demais examinadores e pelo aluno.

Maria Cecilia Pereira de Faria:03008559662

Assinado de forma digital por Maria Cecilia Pereira de Faria:03008559662
Dados: 2022.02.18 16:17:53 -03'00'

Presidente da Banca Examinadora e Coorientadora

Orientador

Examinador 01

Examinador 02

Aluno

Agradecimentos

À minha família, pelo apoio e suporte incondicionais, sem os quais certamente não estaria aqui.

Aos meus amigos dentro e fora da UFMG, aos quais espero ter contribuído com mãos, ombros e ouvidos tão numerosos quanto os que me foram emprestados.

Aos meus orientadores, Dra. Maria Cecília e Dr. Ricardo, pela oportunidade, confiança e ensinamentos.

À UFMG, seu corpo docente e demais profissionais, por me proporcionarem esta jornada.

Muito obrigado.

"We meet in an hour of change and challenge,
in a decade of hope and fear, in an age of both knowledge and ignorance.
The greater our knowledge increases, the greater our ignorance unfolds.
(John F. Kennedy, *We choose to go to the Moon*, 12 de Setembro de 1962)

Resumo

O conceito de *CubeSats* se iniciou em 1999 como uma colaboração entre a *California Polytechnic State University* e a *Stanford University*, e hoje é uma plataforma acessível e amplamente utilizada por instituições de ensino e pelo setor privado para o desenvolvimento de satélites de pequeno porte. Como estes satélites de pequeno porte não são tripulados, o sistema de Telemetria Rastreio e Telecomando (TT&C) consiste em um dos sistemas mais críticos para a missão, uma vez que sua falha resulta em seu fim prematuro.

Este trabalho apresenta um estudo dos principais conceitos referentes ao sistema de TT&C, especialmente no que se diz respeito à potência e velocidade de transmissão mínimas para garantir a correta e completa transmissão dos dados de *payload*. Posteriormente, utilizando os conceitos apresentados, elabora-se um conjunto de necessidades para seu bom funcionamento e boa integração com os demais sistemas de um *CubeSat*.
Palavras-chave: TT&C. CubeSats. Metodologia de Projeto.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

C&DH	*Command and Data Handling*;
COTS	*Commercial Off the Shelf*;
CSMA	*Carrier Sense Multiple Access*;
EPS	*Electrical Power System*;
FCS	*Frame-Check Sequence*;
FSPL	*Free-Space Path Loss*;
FSW	*Flight Software*;
GMAT	*General Mission Analysis Tool*;
GNC	*Guidance, Navigation and Control*;
GPL3	*GNU General Public License V3.0*;
HPBW	*Half-Power Beamwidth*;
IEEE	*Institute of Electrical and Electronics Engineers*;
IPS	*Integrated Propulsion System*;
ISIS	*Innovative Solutions in Space*;
ISM	*Industrial Scientific and Medical*;
ISS	*International Space Station*;
ITU	*International Telecommunication Union*;
MBSE	*Model-based Systems Engineering*;
NASA	*National Aeronautics and Space Administration*;
PID	*Protocol Identifier*;
PLF	*Polarization Loss Factor*;
PSA	*Payload and Subsystems Avionics*;
SSA	*Small Spacecraft Avionics*;

TCS *Thermal Control System*;

TRL *Technology Readyness Level*;

TT&C *Telemetry, Tracking and Control*;

UFMG Universidade Federal de Minas Gerais;

UHF *Ultra High Frequency*;

VHF *Very High Frequency*;

VSWR *Voltage Standing Wave Ratio.*

Lista de símbolos

$\square_{dB}$	$\square$ expresso em Decibel;
$\square_t$	$\square$ da antena transmissora;
$\square_r$	$\square$ da antena receptora;
$\square_{pf}$	$\square$ no sistema perifocal;
$\square_{tc}$	$\square$ no sistema topocêntrico;
$\Delta\square$	Variação de $\square$;
Γ	Coeficiente de reflexão da tensão nos terminais de entrada;
θ	Ângulo azimutal no sistema de coordenadas esférico;
λ	Comprimento de onda;
μ	Constante geo-gravitacional;
ϕ	Ângulo polar no sistema de coordenadas esférico;
ω	Argumento do perigeu;
Ω	Ascensão reta do nodo ascendente;
A	Azimute no sistema topocêntrico;
a	Semi-eixo maior;
D	Diretividade;
D_0	Diretividade máxima;
e	Excentricidade;
e_0	Eficiência total;
e_c	Eficiência de condução;
e_{cd}	Eficiência de radiação;
e_d	Eficiência de dielétrico;
e_r	Eficiência de reflexão;

f	Anomalia verdadeira;
G	Ganho;
G_0	Ganho máximo;
h	Elevação no sistema topocêntrico;
i	Inclinação;
lat	Latitude;
lon	Longitude;
$\hat{p}$	Vetor unitário da polarização do campo elétrico;
p	AX-25 *p-persistence value*;
P_{in}	Potência entregue;
P_r	Potência recebida;
P_{rad}	Potência irradiada;
P_t	Potência transmitida;
$\vec{r}$	Vetor posição;
$\vec{r}_{gs}$	Vetor posição do segmento solo (*ground station*);
R_e	Raio da Terra;
t	Tempo;
t_0	Instante de referência;
T_{102}	AX-25 *p-persistent slot time timer*;
T_{103}	AX-25 *transmitter startup timer*;
T_2	AX-25 *response delay timer*;
T_{CS}	Tempo para se iniciar a transmissão;
u	Anomalia excêntrica;
U	Intensidade de radiação;
Z_0	Impedância da linha de transmissão;
Z_{in}	Impedância de entrada.

Sumário

1 Introdução

Iniciado em 1999 como um projeto colaborativo entre a *California Polytechnic State University* e a *Stanford University* que busca reduzir o tempo de desenvolvimento e possibilitar lançamentos frequentes (MEHRPARVAR et al., 2014), os *CubeSat* tornaram o espaço mais acessível, principalmente para instituições de ensino, uma vez que o tempo de desenvolvimento reduzido permite que os alunos acompanhem todo o processo, do projeto ao lançamento e operação.

Com 1553 *CubeSats* lançados até Abril de 2021 (KULU, 2021), sendo 2 destes interplanetários (KLESH et al., 2018), uma das principais vantagens da plataforma é sua proposta de uniformização e padronização de componentes por meio de unidades "U", cubos de 10 cm de aresta com até 1.33 kg de massa[1], sob os quais os diversos sistemas devem ser acomodados e integrados.

Quanto aos sistemas a serem integrados em múltiplos destas unidades, NASA (2020b) os discretiza em:

- *Electrical Power System* (EPS);
- *Integrated Propulsion System* (IPS);
- *Guidance, Navigation and Control* (GNC);
- *Structures, Materials and Mechanisms*;
- *Thermal Control System* (TCS);
- *Small Spacecraft Avionics*[2] (SSA);
- *Tracking, Telemetry and Control* (TT&C).

Dada uma missão espacial não tripulada, o sistema de TT&C é certamente um dos mais críticos dentre os citados, uma vez que este serve de interface entre o segmento solo e segmento voo, e uma falha neste sistema impossibilita a coleta de telemetria e dados de *payload* e o envio de comandos ao satélite, resultando no fim prematuro da missão, destacando a importância de um projeto de um sistema de TT&C robusto em aplicações como *CubeSats*.

[1] Em sua última revisão do *CubeSat Design Specification* (JOHNSTONE et al., 2020), atualmente em *draft*, o limite de massa foi estendido para 2 kg por unidade.

[2] NASA (2020b) combina os sistemas de *Command and Data Handling* (C&DH), *Flight Software* (FSW), e sistemas menores associados como o de *Payload and Subsystems Avionics* (PSA) no sistema de SSA.

1.1 Motivação

Atualmente a UFMG se encontra em processo de desenvolvimento de seu primeiro *CubeSat*, o PdQSat-1, e portanto é necessário levantar as necessidades de cada sistema para prosseguir com o projeto via *Model-based Systems Engineering* (MBSE).

Além disso, dado que este é o primeiro *CubeSat* da universidade, é vantajoso aprofundar os conhecimentos em todos os sistemas, especialmente nos menos estudados na graduação em Engenharia Aeroespacial da UFMG.

1.2 Justificativa

O sistema de TT&C em sí não é estudado de forma aprofundada na graduação em Engenharia Aeroespacial da UFMG, uma vez que alguns dos conceitos relevantes para seu projeto são mais próximos da Engenharia Elétrica. Ainda assim, o projeto deste sistema é fortemente ditado pelos desafios e peculiaridades da operação no espaço, que por sua vez não é abordada na graduação em Engenharia Elétrica.

Com isso, agrupar os conceitos de ambas as áreas de forma acessível permite que alunos de graduação de ambas engenharias sejam capazes de projetar um sistema de TT&C robusto, o que pode ser uma ferramenta útil para implementações em competições e projetos futuros, como por exemplo do PdQSat-1.

1.3 Objetivos

Subdividindo os objetivos em geral e específico, tem-se:

1.3.1 Objetivo geral

O objetivo geral do presente trabalho é obter relações analíticas ou métodos numéricos representativos da performance de um sistema de TT&C.

1.3.2 Objetivo específico

O objetivo específico do presente trabajo é, utilizando as relações e métodos obtidos no objetivo geral, elaborar necessidades racionais a nível de graduação para garantir o funcionamento e integração de um sistema de TT&C de *CubeSats*.

1.4 Organização do trabalho

O presente trabalho está dividido nos seguintes capítulos:

- **Capítulo 1:** Dedicado à contextualização do trabalho, definição de seus objetivos, bem como à motivação e justificativa para seu desenvolvimento;
- **Capítulo 2:** Dedicado à exposição de conceitos necessários para o desenvolvimento deste trabalho, bem como ao levantamento na literatura do estado da arte dos sistemas de TT&C aplicados a *CubeSats*;
- **Capítulo 3:** Dedicado à descrição da metodologia a ser adotada;
- **Capítulo 4:** Dedicado à exposição das relações analíticas e métodos numéricos, bem como das necessidades de baixo níveis obtidas resultantes;
- **Capítulo 5:** Dedicado à conclusão e sugestões de trabalhos futuros.

2 Revisão bibliográfica

Para o desenvolvimento de um sistema de TT&C, é necessário primeiro entender as principais características dos componentes que formam o sistema, bem como quais componentes estão disponíveis no mercado.

Portanto, esta revisão bibliográfica é dividida em 5 partes. As Seções 2.1 e 2.2 são dedicadas à exposição dos principais conceitos e características relacionadas ao sistema de TT&C, sendo a primeira para conceitos de telecomunicações, e a segunda para conceitos de mecânica orbital. A Seção 2.3 é dedicada ao atual estado da arte dos componentes disponíveis no mercado e a Seção 2.4 é dedicada à implementações similares em missões bem sucedidas disponíveis na literatura. Por fim, uma tabela comparativa com os componentes mais promissores é elaborada na Seção 2.5.

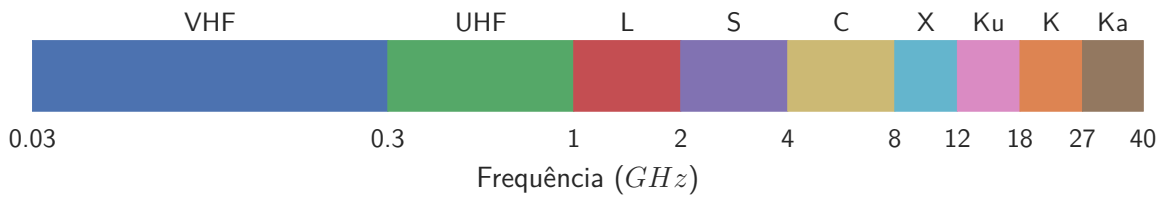
2.1 Conceitos de telecomunicação

As exposições abaixo referentes a antenas foram basadas em Balanis (2016) e NASA (2020b), já as referentes ao protocolo AX-25 foram baseadas em Beech, Nielsen e Noo (1998) e Zielinski (2009). Demais autores são citados conforme necessário.

2.1.1 Bandas de frequência

Os sistemas de TT&C de satélites são fortemente baseados em ondas de rádio com modulação em frequência operando nas faixas entre 30 MHz e 40 GHz. A *International Telecommunication Union* (ITU) nomeia as frequencias do espectro eletromagnético de acordo com a ordem de grandeza do comprimento de onda, com faixas igualmente espaçadas na escala logarítmica (ITU, 2015). Este trabalho, no entanto, utiliza a nomenclatura da *Institute of Electrical and Electronics Engineers* (IEEE) (BRUDER et al., 2019), utilizada na Região 2 da ITU (Américas e Groenlândia), e ilustrada na Figura 1.

Figura 1 – Nomenclatura das bandas de frequência segundo a IEEE



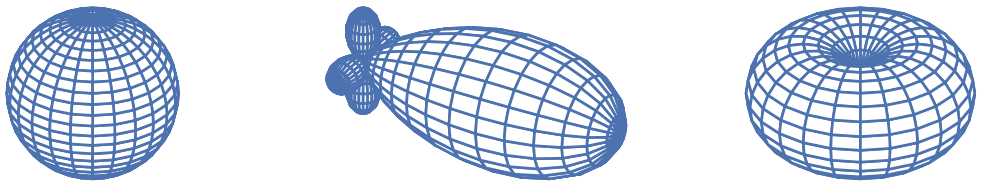
2.1.2 Padrão de radiação

O padrão de radiação de uma antena é definido como a função ou representação gráfica das propriedades de radiação de uma antena. A propriedade representada pode ser intensidade de radiação, diretividade, fase, polarização, entre outros.

Um lobo ou lóbulo de radiação corresponde a uma região do padrão de radiação cercada de regiões com intensidade de radiação relativamente mais fracas. A separação angular entre dois pontos de mesma intensidade em um lobo é chamada de largura de feixe, dos quais os dois mais utilizados são a largura de feixe de meia potência (do inglês *Half-Power Beamwidth*), referente aos pontos com 50% ($\approx -3\,\mathrm{dB}$) da intensidade máxima, e a largura do primeiro feixe nulo (do inglês *First-Null Beamwidth*), referente aos pontos com intensidade nula.

Um padrão de radiação isotrópico apresenta intensidade de radiação homogênea em todas as direções. Apesar de fisicamente impossível, é utilizado amplamente como referência para comparação ou normalização de valores. Já um padrão direcional concentra a intensidade de radiação em uma região em detrimento das outras. Por fim, o padrão omnidirecional distribui a intensidade de radiação uniformemente ao longo de um plano. Estes 3 padrões estão ilustrados na Figura 2.

Figura 2 – Padrões de radiação (a) isotrópico, (b) direcional e (c) omnidirecional



(a) (b) (c)

2.1.3 Diretividade

A diretividade é definida como a razão entre a intensidade de radiação em uma dada direção em relação à média em todas as direções. Quando a direção não é especificada, a direção de intensidade de radiação máxima fica implícita.

Desta forma a diretividade de uma fonte não isotrópica equivale à razão entre a sua intensidade de radiação (U) e a de uma fonte isotrópica em uma dada direção.

$$D(\theta,\phi) = \frac{4\pi U(\theta,\phi)}{\int_0^{2\pi}\int_0^{\pi} U(\theta,\phi)\sin(\theta)d\theta d\phi} \tag{2.1}$$

$$D_{dB}(\theta,\phi) = 10\log_{10}(D(\theta,\phi)) \tag{2.2}$$

O uso de uma fonte isotrópica como referência resulta na unidade dBi.

2.1.4 Eficiência da antena

A potência de entrada, fornecida nos terminais da antena, não é a mesma potência irradiada, uma vez que existem perdas na estrutura da antena. A eficiência, portanto, leva em consideração perdas por reflexão causada por diferenças nas impedâncias da linha de transmissão e da antena, bem como perdas por efeito Joule, tanto por condução quanto pelo meio dielétrico:

$$e_0 = e_r e_c e_d \tag{2.3}$$

As eficiências de condução (e_c) e de dielétrico (e_d) são difíceis de se calcular, porém podem ser obtidas experimentalmente. Como as duas eficiências não podem ser separadas, são frequentemente expressadas em conjunto, como eficiência de radiação (e_{cd}).

Já a eficiência de reflexão é obtida em função do coeficiente de reflexão da tensão nos terminais de entrada da antena, que é dado por:

$$\Gamma = \frac{Z_{in} - Z_0}{Z_{in} + Z_0} \tag{2.4}$$

Do coeficiente de reflexão da tensão é possível definir outro parâmetro de interesse, a razão da onda estacionária de tensão (do inglês *Voltage Standing Wave Ratio*):

$$VSWR = \frac{1 + |\Gamma|}{1 - |\Gamma|} \tag{2.5}$$

Por fim, a eficiência de reflexão é dada por:

$$e_r = 1 - |\Gamma|^2 = 1 - \left(\frac{VSWR - 1}{VSWR + 1}\right)^2 \tag{2.6}$$

2.1.5 Ganho

O ganho é definido como a razão entre a intensidade de radiação em um dada direção em relação a uma fonte isotrópica sem perdas de mesma potência de entrada. Quando a direção não é especificada, a direção de ganho máximo fica implícita. Desta forma, sua definição é próxima da de diretividade, com a distinção que a diretividade usa a potência irradiada (P_{rad}), enquanto o ganho usa a potência de entrada (P_{in}).

Seguindo as definições da IEEE, o ganho não considera perdas por reflexão, portanto:

$$P_{rad} = e_{cd} P_{in} \tag{2.7}$$

$$G(\theta, \phi) = e_{cd} D(\theta, \phi) \tag{2.8}$$

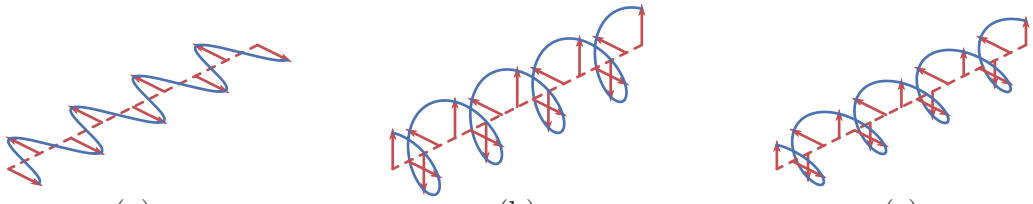
$$G_{dB}(\theta, \phi) = 10 \log_{10}(G(\theta, \phi)) = D_{dB}(\theta, \phi) + 10 \log_{10}(e_{cd}) \tag{2.9}$$

2.1.6 Polarização

A polarização de uma antena é definida como a polarização da onda irradiada por ela. A polarização da onda irradiada, por sua vez, é uma propriedade que descreve a direção e magnitude do vetor campo elétrico ao longo do tempo. Como a energia irradiada varia com a direção, a polarização também pode variar. Desta forma, quando a direção não é especificada, a direção de ganho máximo fica implícita.

A polarização pode ser classificada em linear, circular ou elíptica, conforme ilustrado na Figura 3.

Figura 3 – Polarizações (a) linear, (b) circular e (c) elíptica



(a) (b) (c)

Quando a polarização da antena receptora ($\hat{p}_r$) não é a mesma da antena transmissora ($\hat{p}_t$), há um descasamento de polarização, e a potência recebida não é máxima. Esta perda é expressa pelo Fator de perda por polarização (do inglês *Polarization Loss Factor*):

$$PLF = |\hat{\rho}_t \cdot \hat{\rho}_r|^2 \tag{2.10}$$

2.1.7 Sensibilidade do receptor

A sensibilidade é a menor potência para qual o receptor é capaz de claramente distinguir o sinal transmitido do ruido de fundo, isto é, a menor potência possível para que a transmissão de dados seja possível.

Esta potência é medida nos terminais do receptor, e frequentemente é normalizada utilizando 1 mW como referência, resultando em dBm como unidade.

2.1.8 Equação de Friis

A equação de Friis relaciona as potências transmitida e recebida entre duas antenas no espaço livre:

$$\frac{P_r}{P_t} = e_{cd,t} e_{cd,r} \left(\frac{\lambda}{4\pi r}\right)^2 D_t(\theta_t, \phi_t) D_r(\theta_r, \phi_r) \tag{2.11}$$

A Equação 2.11 assume antenas com casamento de impedâncias e polarizações compatíveis. Ao se incluir as eficiências reflectivas de ambas antenas e o fator de perda por polarização, a equação se torna:

$$\frac{P_r}{P_t} = e_{cd,t}e_{cd,r}(1 - |\Gamma_t|^2)(1 - |\Gamma_r|^2)\left(\frac{\lambda}{4\pi r}\right)^2 D_t(\theta_t, \phi_t)D_r(\theta_r, \phi_r)|\hat{\rho}_t \cdot \hat{\rho}_r|^2 \tag{2.12}$$

Já para antenas com casamento de impedâncias e polarizações compatíveis alinhadas na direção de intensidade de radiação máxima, a Equação 2.11 pode ser simplificada para:

$$\frac{P_r}{P_t} = \left(\frac{\lambda}{4\pi r}\right)^2 G_{0,t}G_{0,r} \tag{2.13}$$

2.1.8.1 Fator de perda do espaço livre

O termo $(\lambda/4\pi r)^2$ é chamado de fator de perda do espaço livre (do inglês *free-space loss factor* ou *free-space path loss*), e representa a atenuação geométrica causada pela distribuição esférica da energia emitida pela antena transmissora. Para transformar o fator de adimensional para Decibel, tem-se:

$$FSPL_{dB} = 20\log_{10}\left(\frac{\lambda}{4\pi r}\right) \tag{2.14}$$

2.1.8.2 Link budget

Os coeficientes das Equações 2.11, 2.12 e 2.13 estão expressos de forma adimensional. Desta forma, isolando a potência recebida (P_r) na Equação 2.11, e convertendo os coeficientes para Decibel, tem-se:

$$P_{r,dB} = P_{t,dB} + G_{t,dB}(\theta_t, \phi_t) + G_{r,dB}(\theta_r, \phi_r) + 20\log_{10}\left(\frac{\lambda}{4\pi r}\right) \tag{2.15}$$

Que pode ser genericamente interpretada como:

$$P_{r,dB} = P_{t,dB} + Ganhos_{dB} - Perdas_{dB} \tag{2.16}$$

2.1.9 Protocolo AX-25

Um protocolo de comunicação define as regras e procedimentos para a correta transmissão de informação. O principal protocolo suportado pelos rádios levantados neste trabalho é o AX-25, um protocolo da camada de enlace de dados (*data link layer* ou camada 2) para operações radioamadoras (BEECH; NIELSEN; NOO, 1998).

2.1.9.1 Estrutura

Os dados são organizados e enviados em pequenos blocos de dados, chamados *frames*. A Tabela 1 apresenta os campos de um *information frame* AX-25.

Tabela 1 – Estrutura de um frame AX-25

Flag	Address	Control	PID	Information	FCS	Flag
1 Byte	14 Bytes	1 Byte	1 Byte	0 - 256 Bytes	2 Bytes	1 Byte

Fonte: Adaptado de Beech, Nielsen e Noo (1998)

A flag `01111110` indica o início e final de cada *frame*, e não pode ocorrer em nenhum outro lugar, o que é garantido por meio de *bit stuffing*.

O campo *address* identifica a origem e destino do *frame*.

O campo *control* identifica o tipo e número do *frame*, e permite o envio de até 7 *frames* consecutivos por vez.

O campo *information* contém os dados a serem transmitidos, caso utilize algum protocolo da camada de rede (*network layer* ou camada 3), este é identificado pelo campo *Protocol Identifier* (PID).

Por fim, o *Frame-Check Sequence* (FCS) é um valor calculado por ambos os lados, e garante que o *frame* não foi corrompido. O protocolo não suporta *forward error correction*, sendo necessário reenviar *frames* corrompidos.

2.1.9.2 *Bit stuffing*

Para impedir que a flag `01111110` ocorra ao longo do conteúdo de um *frame*, ao ser montado para transmissão, um bit `0` é inserido após qualquer sequência de 5 bits `1` consecutivos. Ao ser recebido os dados são restaurados removendo todo bit `0` após uma sequência de 5 bits `1`. Desta forma, dependendo de seu conteúdo, o *frame* físico pode ser até 20 % mais longo do que a soma dos tamanhos de cada campo.

O modelo descrito em Zielinski (2009) usa um fator de $\frac{63}{62}$ aplicado ao tamanho total do *frame* como valor médio para o efeito de *bit stuffing*.

2.1.9.3 *Carrier Sense Multiple Access*

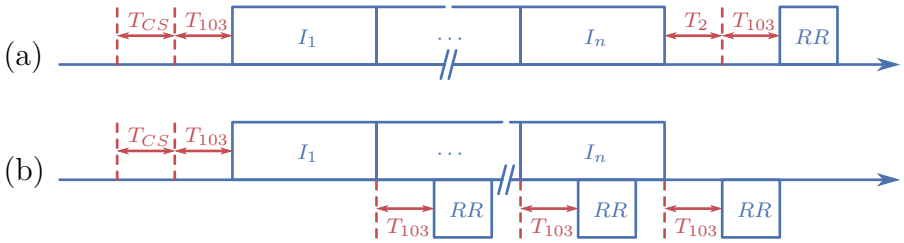
Carrier Sense Multiple Access (CSMA) é um protocolo de controle de acesso ao meio em que um nodo verifica o *status* do canal antes de transmitir. A variação *p-persistant* visa reduzir colisões quando dois nodos detectam o canal livre, uma vez que cada nodo apresenta uma probabilidade ($\frac{p+1}{256}$ no protocolo AX-25) de se iniciar a transmissão a cada *time slot* (T_{102} no protocolo AX-25) com o canal livre. Desta forma, o tempo para se iniciar a transmissão (T_{CS}) varia a cada ciclo, e sua média para o protocolo AX-25 considerando um canal completamente livre é:

$$\bar{T}_{CS} = \frac{256}{p+1}\frac{T_{102}}{2} \tag{2.17}$$

2.1.9.4 Sequência de operações

Antes de iniciar um ciclo de transmissão, o transmissor verifica se o canal está livre por meio de *p-persistant* CSMA. Com o canal livre, e após T_{103}, os *information frames* são enviados. Para um *link half-duplex*[1], após a recepção do último *frame*, obtido após T_2 sem o envio de um novo *frame*, o receptor aguarda T_{103} e envia um *receiver ready frame* completando o ciclo. Já no caso de um *link full-duplex*[2], uma resposta pode ser enviada para cada *information frame* imediatamente após ser recebida, resultando em um ciclo mais curto. A Figura 4 ilustra ambos os ciclos.

Figura 4 – Ciclo de transmissão do protocolo AX-25 para *links* (a) *half-duplex* e (b) *full-duplex*



Fonte: Adaptado de Zielinski (2009)

2.2 Conceitos de mecânica orbital

As exposições abaixo foram basadas em Kuga, Kondapalli e Carrara (2012), Curtis (2005) e Chobotov (2002). Demais autores são citados conforme necessário.

2.2.1 Sistemas de coordenadas

Existem diversas sistemas de coordenadas aplicados no estudo do movimento de corpos celestes, cada um com suas vantagens e desvantagens. Nesta seção estão explicitados apenas os sistemas utilizados no desenvolvimento deste trabalho, ilustrados na Figura 5.

2.2.1.1 Sistema topocêntrico

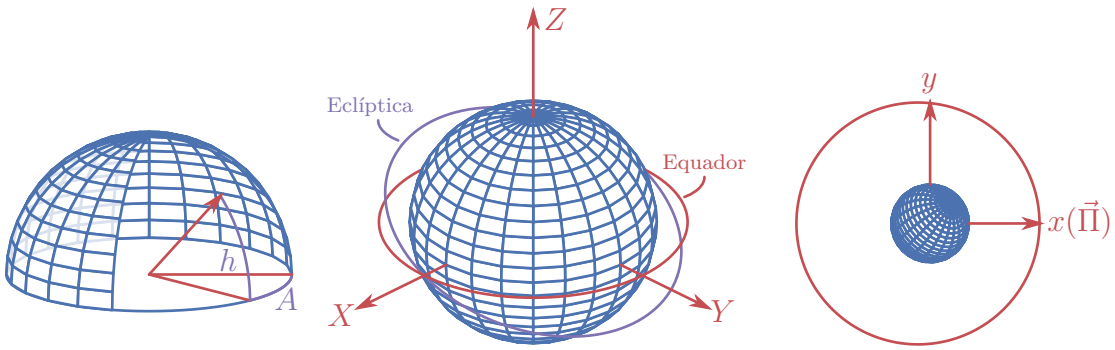
Sistema de coordenadas esférico centrado em um ponto na superfície da Terra com o horizonte local como plano fundamental, define dois ângulos para se localizar objetos na esfera celeste:

- **Elevação (h):** Ângulo entre o plano do horizonte e objeto;

[1] Em um *link half-duplex*, ambos os transceptores compartilham um canal, e apenas um pode transmitir informação por vez, sendo necessário coordenação para evitar interferência.

[2] Em um *link full-duplex*, cada transceptor utiliza um canal dedicado, de forma que ambos podem transmitir informação simultaneamente sem causar interferência.

Figura 5 – Sistemas de coordenadas (a) topocêntrico, (b) geocêntrico inercial e (c) perifocal



(a) (b) (c)

- **Azimute (A):** Ângulo em torno do horizonte entre o objeto e o Norte Geográfico, com o ângulo crescendo para o Leste.

  Note que esta definição resulta em um sistema levogiro.

2.2.1.2 Sistema geocêntrico inercial

Sistema de coordenadas cartesiano centrado no centro de massa da Terra com o plano equatorial como plano de referência, apresenta os eixos:

- **Eixo X:** Equinócio vernal[3];
- **Eixo Y:** Completa o sistema dextrogiro;
- **Eixo Z:** Norte Geográfico.

2.2.1.3 Sistema perifocal

Sistema de coordenadas cartesiano centrado no foco da órbita, apresenta os eixos:

- **Eixo x:** Perigeu[4] ($\vec{\Pi}$);
- **Eixo y:** Completa o sistema dextrogiro;
- **Eixo z:** Momento angular do corpo de interesse.

2.2.2 Problema de 2 corpos

O problema de 2 corpos consiste em descrever a trajetória de dois corpos no espaço submetido apenas às forças gravitacionais mútuas. Apresenta solução analítica, com órbitas no formato de cônicas.

[3] O equinócio vernal é definido como o vetor no espaço resultante da interseção do plano equatorial com a eclíptica que, no equinócio de primavera, aponta para o Sol.

[4] Para sistemas em que a Terra não é o corpo central, utiliza-se o termo periapse.

Dada a missão típica de *CubeSats*, este trabalho considera apenas as soluções do problema de 2 corpos com órbitas fechadas ($0 \leq e < 1$), além disso, por comodidade, as simulações neste trabalho são realizadas no domínio da anomalia excêntrica (u), uma vez que desta forma o tempo é obtido de forma direta, o que não é verdade no domínio do tempo (t), em que as anomalias são obtidas solucionando uma equação transcendental:

$$\sqrt{\frac{\mu}{a^3}}(t - t_0) = u - e\sin(u) \tag{2.18}$$

A anomalia excêntrica pode ser obtida da anomalia verdadeira pela relação:

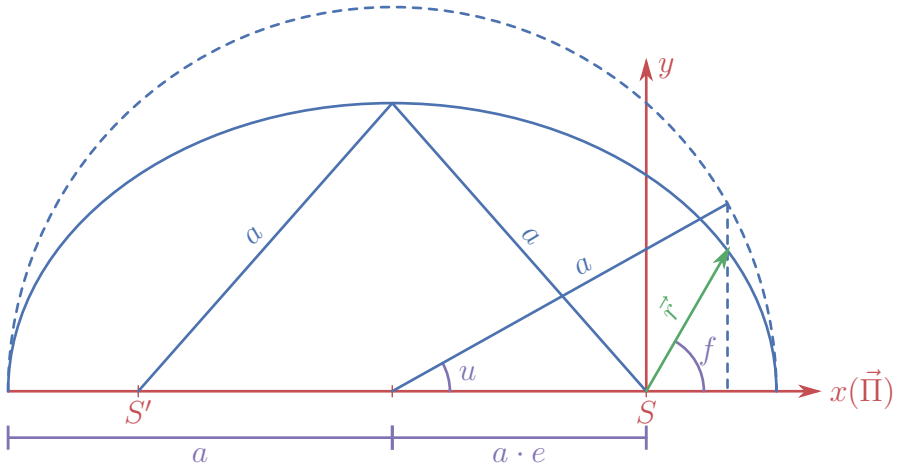
$$\tan^2\left(\frac{f}{2}\right) = \frac{1+e}{1-e}\tan^2\left(\frac{u}{2}\right) \tag{2.19}$$

O vetor posição para uma dada anomalia excêntrica é, então, dado por:

$$\vec{r}_{pf} = a \cdot \begin{bmatrix} \cos(u) - e \\ \sin(u)\sqrt{1-e^2} \\ 0 \end{bmatrix} \tag{2.20}$$

A Figura 6 ilustra os parâmetros das Equações 2.19 e 2.20 no plano perifocal de uma órbita elíptica.

Figura 6 – Movimento elíptico



Fonte: Adaptado de Kuga, Kondapalli e Carrara (2012)

2.2.3 Elementos Keplerianos

Como visto na Seção 2.2.2, é possível descrever uma órbita no plano com apenas dois parâmetros, e descrever um ponto com um terceiro parâmetro, porém, para posicionar esta órbita no espaço é necessário outros 3 parâmetros. Estes 6 parâmetros definem

completamente a órbita e posição de um objeto no espaço, e são normalmente chamado de elementos Keplerianos, elementos orbitais ou vetor estado:

- **Semi-eixo maior (a):** Especifica o tamanho da cônica;
- **Excentricidade (e):** Especifica o formato da cônica;
- **Anomalia verdadeira (f):** Especifica a posição do corpo na cônica;
- **Inclinação (i):** Ângulo entre o plano da órbita e o plano de referência;
- **Ascensão reta do nodo ascendente (Ω):** Ângulo entre o equinócio vernal (X) e o nodo ascendente ($\vec{\Omega}$);
- **Argumento do perigeu (ω):** Ângulo entre o perigeu ($\vec{\Pi}$) e o nodo ascendente ($\vec{\Omega}$).

A anomalia verdadeira é frequentemente substituída pela anomalia média (M), e o semi-eixo maior pelo momento angular (h). O nodo ascendente corresponde à posição da órbita que cruza o plano de referência do hemisfério inferior para o superior.

Especificados a posição e órbita do corpo de interesse no sistema perifocal, este pode então ser transformado para o geocêntrico inercial por 3 rotações sucessivas:

$$\vec{r}_{pf} = R(\Omega, i, \omega)\vec{r} \Leftrightarrow \vec{r} = R(\Omega, i, \omega)^{-1}\vec{r}_{pf} \tag{2.21}$$

$$R(\Omega, i, \omega) = R_3(\Omega)R_1(i)R_3(\omega) \tag{2.22}$$

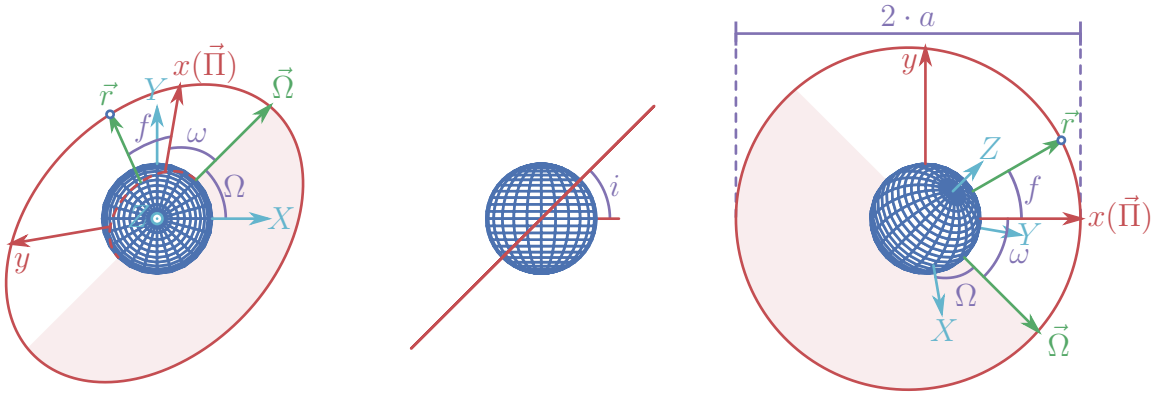
$$R(\Omega, i, \omega) = \begin{bmatrix} \cos(\Omega) & \sin(\Omega) & 0 \\ -\sin(\Omega) & \cos(\Omega) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(i) & \sin(i) \\ 0 & -\sin(i) & \cos(i) \end{bmatrix} \begin{bmatrix} \cos(\omega) & \sin(\omega) & 0 \\ -\sin(\omega) & \cos(\omega) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.23}$$

As rotações, bem como os elementos Keplerianos para uma órbita elíptica, estão ilustrados na Figura 7.

2.3 Estado da arte

A metodologia adotada para este levantamento do estado da arte dos componentes de TT&C consiste em 2 etapas sequenciais. Primeiramente, a partir do histórico de satélites lançados, verifica-se as bandas mais utilizadas, de forma a selecionar-se a banda com maior *flight heritage*. Em seguida, são levantados componentes compatíveis com a banda selecionada e com robustez adequada.

Figura 7 – Elementos Keplerianos

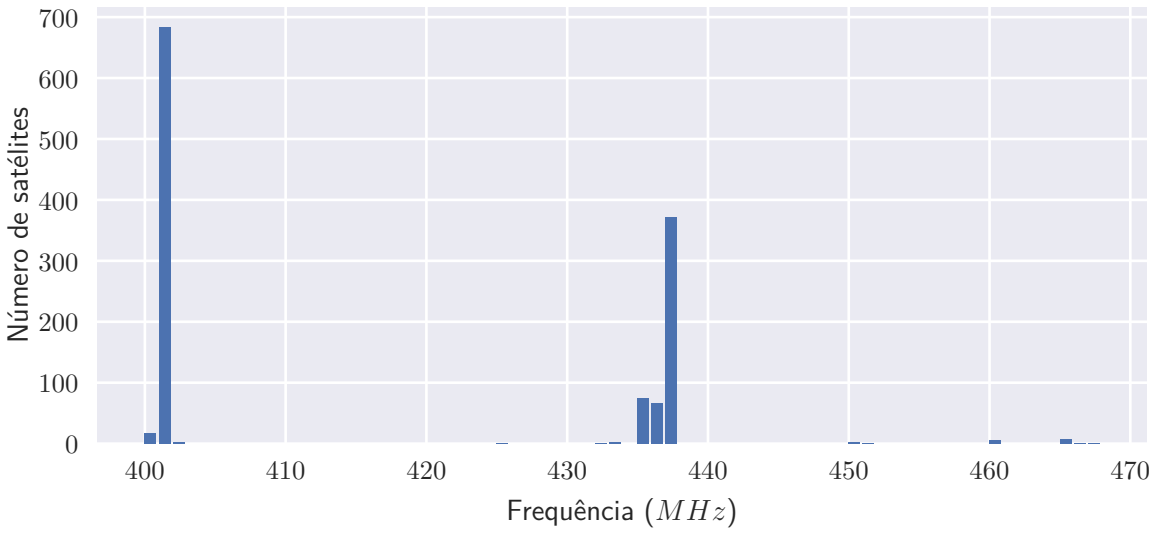


2.3.1 Bandas em uso

Para os levantamento das bandas e frequências mais comuns, utilizou-se Kulu (2021). Apenas as bandas e frequências de *downlink* foram analisadas, uma vez que os dados para *uplink* foram considerados incompletos demais para se obter informações satisfatórias, com apenas 15,6% das entradas no banco de dados apresentando informações sobre a banda de *uplink*, e nenhuma entrada quanto à frequência utilizada.

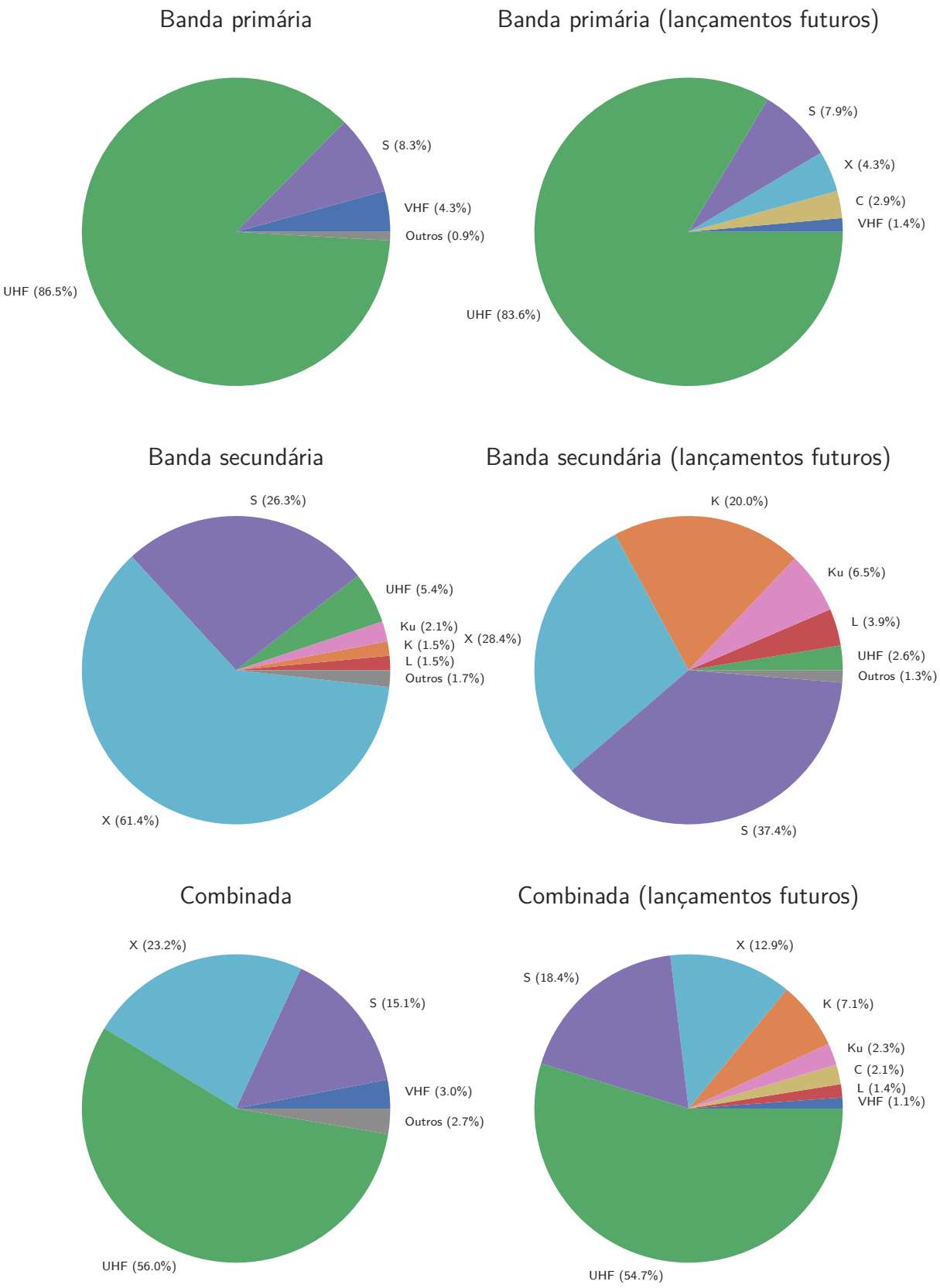
A distribuição das bandas de *downlink* para *nanosats* lançados até o momento estão dispostas na primeira coluna da Figura 9, enquanto as tendência futuras, obtidas das entradas cadastradas em Kulu (2021) com lançamentos ainda pendentes, estão dispostas na segunda coluna. Por fim, as frequências (agrupadas por MHz) mais utilizadas na banda UHF estão dispostas na Figura 8.

Figura 8 – Distribuição das frequências para *downlink* na região da banda UHF mais utilizada



Fonte: Adaptado de Kulu (2021)

Figura 9 – Distribuição das bandas utilizadas para *downlink*



Fonte: Adaptado de Kulu (2021)

Da distribuição de uso das bandas, nota-se a predominância da banda UHF para *downlink*, apesar da tendência de diversificação das bandas no futuro, principalmente das secundárias, o que é condizente com o apresentado em NASA (2020b).

Já da distribuição das frequências, destaca-se as faixas de interesse e suas respectivas atribuições para a Região 2 e Brasil (ANATEL, 2020) a seguir:

- **144 a 146 MHz (VHF):** Radioamador e radioamador por satélite, conforme resolução Anatel nº 697/18 e ato nº 9106/18;
- **401 a 402 MHz (UHF):** Auxílio à meteorologia, operação espacial, exploração da terra por satélite e meteorologia por satélite, conforme resolução Anatel nº 685/17;
- **435 a 438 MHz (UHF):** Radiolocalização, radioamador, radioamador por satélite e exploração da terra por satélite (ativo), conforme resoluções Anatel nº 681/17, 685/17, 697/18 e 716/19 e ato nº 9106/18;
- **902 a 908 e 915 à 928 MHz (UHF):** Fixo, radioamador, móvel (exceto móvel aeronáutico) e radiolocalização, conforme resolução Anatel nº 697/18 e ato nº 9106/18.

Destacam-se também as seguintes notas internacionais aplicáveis às frequências levantadas:

> **5.150** - As seguintes faixas de frequências [...] 902-928 MHz na Região 2 (frequência central 915 MHz) [...] são também destinadas para aplicações industriais, científicas e médicas (ISM). Serviços de radiocomunicações operando nessas faixas de frequências deverão aceitar interferência prejudicial que podem ser causadas por estas aplicações. Equipamentos ISM operando nestas faixas estão sujeitas as disposições do nº 15.13.
>
> **5.282** - O serviço de radioamador por satélite pode operar nas faixas 435-438 MHz [...] sujeito a não causar interferência prejudicial aos outros serviços operando de acordo com a Tabela (ver o nº 5.43). As Administrações ao autorizarem tal uso deverão garantir que qualquer interferência prejudicial causada por emissões oriundas de uma estação do serviço de radioamador por satélite seja imediatamente eliminada, conforme as disposições do nº 25.11. [...]
>
> Fonte: (ANATEL, 2020)

2.3.2 Componentes disponíveis no mercado

Para o levantamento de componentes foi utilizado o banco de dados da *SmallSat Parts On Orbit Now* (NASA, 2021), bem como os componentes listados no catálogo da *CubeSatShop* (ISIS, 2021). Buscaram-se transceptores e antenas que operam na banda UHF com *Technology Readyness Level* (TRL) igual ou superior a 8 (sistemas *mission qualified* ou *mission proven*). Por fim, a lista de componentes obtida foi comparada com os

componentes listados como estado da arte para pequenas espaçonaves (NASA, 2020b), observando-se resultados semelhantes.

Os transceptores escolhidos estão dispostos na Tabela 2 e as antenas na Tabela 3.

Tabela 2 – Tranceptores COTS para banda UHF

Componente	Fabricante	TRL	Preço
Cadet Nanosat Radio	L3 Comm	9	N/C
Full Duplex Transceiver	ISIS	9	€8500
Lithium-1	AstroDev	9	N/C
NanoCom AX100	GomSpace	9	N/C
SpaceQuest TRX-U	SpaceQuest	8	$18000
TOTEM nanosatellite SDR	Alén Space	9	€18000
UHF Transceiver II	EnduroSat	9	$4600
VUTRX	AAC Clyde	9	N/C

Fonte: Adaptado de NASA (2021), ISIS (2021)

Tabela 3 – Antenas COTS para banda UHF

Componente	Fabricante	TRL	Preço
Deployable antenna system (1U/3U)	ISIS	9	€4500
Deployable antenna system (6U/12U)	ISIS	9	N/C
Helios deployable antenna	HCT	9	$36000
NanoCom ANT430	GomSpace	9	N/C
NanoCom ANT-6F	GomSpace	9	N/C
SpaceQuest ANT-100 II	SpaceQuest	8	$1000
UHF Antenna III	EnduroSat	9	$4000

Fonte: Adaptado de NASA (2021), ISIS (2021)

2.4 Implementação em outros trabalhos

Por fim, obteve-se informações sobre as implementações do subsistema de TT&C de outros projetos, dando ênfase para missões bem sucedidas, de forma a beneficiar componentes robustos. Também foi dada devida atenção a projetos brasileiros, uma vez que estes compartilham dos mesmos desafios quanto à disponibilidade e importação de componentes.

2.4.1 NanosatC-BR

Lançado em 19 de Junho de 2014, o NanosatC-BR1 é primeiro nanossatélite científico brasileiro a ser colocado em órbita. Este usa a plataforma 1U da ISIS, com um transceptor TRXUV (COSTA et al., 2017) operando na frequência de 145.86 MHz. Em 22 de março de 2021 a constelação é entendida com o NanosatC-BR2, desta vez utilizando uma configuração 2U e um transceptor *VHF uplink/UHF downlink Full Duplex Transceiver* também da ISIS (ALMEIDA; MATTIELLO-FRANCISCO, 2017) operando em uma frequência não especificada.

Atualmente estuda-se a possibilidade de se reutilizar os modelos de engenharia dos NanosatC-BR1 e NanosatC-BR2 como modelos de voo com o desenvolvimento de novas cargas úteis para as missões NanosatC-BR3 e NanosatC-BR4 (SCHUCH et al., 2019).

2.4.2 ITASAT

Apresenta dois canais de *downlink*. O primeiro na banda VHF, sendo operado pelo transceptor TRXUV da ISIS na frequência de 145.86 MHz e o segundo, para dados de *payload* mais pesados, operando por um rádio S-band TXS também da ISIS na frequência de 2.4 GHz. Por fim, o *uplink*, na banda UHF, é operado pelo mesmo transceptor TRXUV do *downlink* (SATO et al., 2019).

2.4.3 FloripaSat

Ambos o *hardware* e *software* foram desenvolvidos *in-house*, porém são *open-source*, sendo distribuídos por meio da *GNU General Public License V3.0* (GPL3), disponíveis em Bezerra et al. (2019). Utiliza os protocolos NGHam e AX.25 nas frequências de 436.1 MHz para *downlink* e *uplink* e 145.9 MHz para *beacon*.

2.4.4 CONASAT

Surgiu como projeto após o sucesso do NanosatC-BR1, porém não tem atualizações desde 2017 (INPE, 2017). Utilizaria o transceptor *VHF uplink/UHF downlink Full Duplex Transceiver* da ISIS (ALVES et al., 2019), e operaria nas faixas de frequências entre 145.8 e 146 MHz para *uplink* e entre 435 e 438 MHz para *downlink*.

2.4.5 CRON-1 / nanoMIRAX

Primeiro projeto de nanossatélite privado no Brasil. Ainda não lançado, utilizará um transceptor TRXUV da ISIS (BRAGA et al., 2020) operando nas faixa de 130 a 160 MHz para *uplink* e 482 a 486 MHz para *downlink*.

2.5 Métodos Comparativos

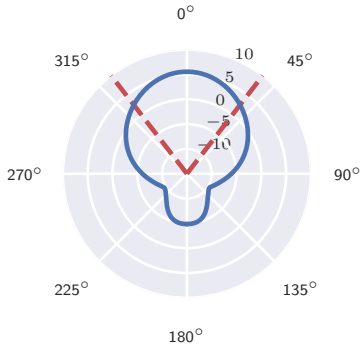
A partir dos componentes levantados na Seção 2.3.2 e das implementações discutidas na Seção 2.4, selecionaram-se componentes de interesse para elaboração de uma tabela comparativa. A Tabela 4 reúne os dados para transceptores, e a Tabela 5 para antenas. Para o transceptor do FloripaSat foi necessário assumir os valores dispostos no *datasheet* do IC utilizado, também assumiu-se uma modulação com um *bit* por símbolo para converter de *symbol per second* (sbs) para *bit per second* (bps).

Tabela 4 – Comparação dos tranceptores selecionados

		EnduroSat	ISIS	FloripaSat
Tipo		*Half-duplex*	*Full-duplex*	*Half-duplex*
Frequência Rx	(MHz)	430 a 440	145.8 a 146	410 a 450
Frequência Tx	(MHz)	430 a 440	435 a 438	410 a 450
Potência de transmissão	(dbm)	30	27	30
Sensibilidade	(dbm)	−121	−104	−126
Taxa de transmissão	(bps)	4800 a 19600	1200 a 9600	2400
Massa	(g)	90	75	72
Temperatura	(°C)	−35 a 80	−20 a 60	N/C

Fonte: Adaptado de EnduroSat (2021b), ISIS (2016b), Bezerra et al. (2019), NiceRF (2018)

Tabela 5 – Comparação das antenas selecionadas

		EnduroSat	ISIS
Configuração		N/C	Monopolo Dipolo Turnstile
Polarização		Circular	Circular Linear
VSWR		N/C	1.19 : 1
Diagrama de Radiação		N/C	
Massa	(g)	85	77 a 85
Temperatura	(°C)	−40 a 125	−20 a 60

Fonte: Adaptado de EnduroSat (2021a), ISIS (2016a), ISIS (2020)

3 Metodologia

O desenvolvimento de um *CubeSat* é um processo multidisciplinar, e requer uma boa integração entre todos os seus subsistemas. Desta forma, um bom projeto de um subsistema não só garante seu bom funcionamento, como também visa uma boa integração com os demais subsistemas. Este trabalho busca atingir estes objetivos elaborando um conjunto de necessidades tanto para o sistema de TT&C quanto para os demais sistemas relacionados.

Para se elaborar este conjunto de necessidades, é necessário primeiro enunciar o problema a ser resolvido:

> O sistema de TT&C deve ser capaz de transmitir os dados de *payload* ao segmento solo a cada janela de comunicação.

A partir das informações levantadas no Capítulo 2, o enunciado do problema pode então ser derivado em duas necessidades funcionais[1] de alto nível:

1. O sistema deve transmitir potência suficiente para sobrepor as perdas até o segmento solo;
2. O sistema deve transmitir os dados com velocidade suficiente para completar a transmissão em uma janela de comunicação.

Para cada necessidade de alto nível, levantam-se as variáveis de interesse e sistemas relacionados, na qual busca-se derivar uma relação analítica ou método numérico que represente a relação entre as variáveis. Esta é então usada para se desdobrar um conjunto de necessidades de baixo nível que prova racionalmente a conformidade com a necessidade de alto nível desdobrada. Posteriormente estas necessidades são utilizadas para se elaborar uma solução candidata, que é então formalizada pelos requisitos.

No caso das necessidades de alto nível descritas acima, as variáveis de interesse consistem na distância entre antena transmissora e receptora, nos ângulos de visada, na velocidade angular para rastreio, e na duração da janela de comunicação, que dependem dos parâmetros orbitais e da atitude do *CubeSat* regidos pelo projeto de órbita e pelo sistema de aquisição e controle de atitude respectivamente. As relações analíticas e métodos numéricos para obtenção da potência mínima e tamanho dos dados de *payload* máximo para transmissão em função destas variáveis e de parâmetros dos sistemas relacionados são obtidos no Capítulo 4.

[1] O caráter funcional se dá por especificar o que deve ser feito, porém não como.

4 Resultados

O intuito deste capítulo é explicitar as principais variáveis de interesse, seus efeitos e relações com o subsistema de TT&C e demais decisões de projeto, bem como prover argumentos racionais para a elaboração de requisitos capazes de garantir a correta transmissão dos dados da *payload* considerando os componentes disponíveis no mercado e características dos demais subsistemas.

Além disso, caso necessário, as relações aqui explicitadas servem de ferramenta para análise de possíveis soluções compromisso.

4.1 Valores arbitrados para simulação

Para algumas da análises das próximas seções, é necessário arbitrar valores para o sistema, portanto, valores típicos baseados nos dados obtidos no Capítulo 2 foram selecionados. Exceto quando explicitamente descrito, os valores a seguir são utilizados.

4.1.1 Antenas e Rádios

A antena e rádio transmissores foram baseados nos valores da Tabelas 5 e 4, enquanto a antena e rádio receptores foram baseados na *Ground Station* da ISIS (2019).

Ambas antenas foram modeladas com padrões de radiação direcionais, dos quais são obtidas as intensidades de radiação, o ganho e o HPBW por meio de integração numérica e das Equações 2.1 e 2.8.

A antena receptora foi concebida como um lobo do tipo $\cos^n(\theta)$ com domínio $\{\theta \in \mathbb{R} | 0 \leq \theta \leq \pi/2\}$, ganho máximo de 15 dBi e eficiência de radiação unitária. Para este caso, o expoente pode ser obtido analiticamente em função do ganho novamente a partir das Equações 2.1 e 2.8:

$$G_0 = 1 \cdot D_0 = 1 \cdot \frac{4\pi \cdot 1}{\int_0^{2\pi} \int_0^{\pi/2} \cos^n(\theta) \sin(\theta) d\theta d\phi} \Rightarrow n = \frac{G_0}{2} - 1 \tag{4.1}$$

Tem-se então:

$$U_r(\theta, \phi) = \cos(\theta)^{14.8114} \tag{4.2}$$

$$G_{r,dB}(\theta, \phi) = 15 + 10 \log_{10} \left(U_r(\theta, \phi) \right) \tag{4.3}$$

Já a antena transmissora foi concebida com componentes isotrópico, bidirecional e cardioide, bem como uma eficiência de radiação não unitária. Um ajuste de curva com os dados da Tabela 5 foi feito, obtendo-se as equações a seguir:

$$U_t(\theta, \phi) = 0.0317 + 0.0727\cos(\theta)^6 + 0.8956\cos(\theta/2)^{8.9238} \tag{4.4}$$

$$G_{t,dB}(\theta, \phi) = 10\log_{10}\left(\frac{4\pi U_t(\theta, \phi)}{2.5894}\right) - 1.8155 \tag{4.5}$$

Os dados arbitrados estão resumidos nas Tabelas 6 e 7, e o padrão de radiação para ambas antenas está disposto na Figura 10.

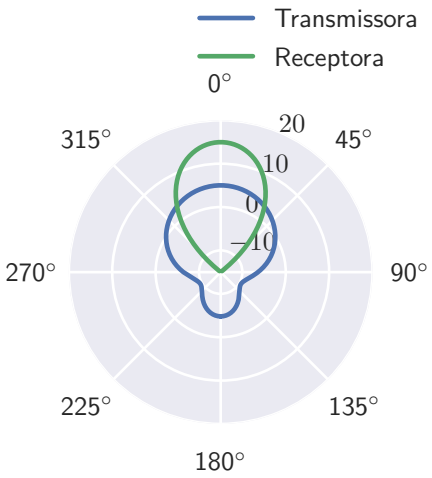
Tabela 6 – Valores arbitrados para os rádios

		Transmissor	Receptor
Tipo		*Half-duplex*	*Half-duplex*
Potência	(dBm)	30	N/A
Sensibilidade	(dBm)	N/A	−120
Bitrate	(bps)	2400	2400
Frequência	(MHz)	437	437

Tabela 7 – Valores arbitrados para as antenas

		Transmissora	Receptora
Ganho	(dBi)	Equação 4.5	Equação 4.3
Ganho máximo	(dBi)	5	15
VSWR		1.20 : 1	1.20 : 1
HPBW	(°)	88.1	35.1
Polarização		RHCP	RHCP

Figura 10 – Diagrama de radiação para as antenas arbitradas



4.1.2 Parâmetros orbitais

Para a seleção, buscou-se abranger tanto órbitas de missões típicas de *CubeSats* quanto órbitas com parâmetros extremos capazes de demonstrar os efeitos de outras variáveis de projeto na janela de comunicação.

A ISS, por sua capacidade de lançar *CubeSats*, foi escolhida como uma órbita quasicircular típica. Órbitas heliossíncronas também são bastante úteis dependendo da missão, especialmente a de 15 revoluções por dia por conta de sua altitude. Por fim, selecionou-se uma órbita polar de mesma altitude da heliossíncrona, para demonstrar o efeito da latitude do segmento solo, e uma órbita Molnya para demonstrar o efeito da excentricidade e altitude.

Os parâmetros estão resumidos na Tabela 8.

Tabela 8 – Parâmetros orbitais arbitrados

		ISS	Heliossíncrona	Polar	Molnya
a	(km)	6798.5	6932.4	6932.4	26 561.8
e		0.0004	0.0000	0.0000	0.7370
i	(°)	51.6	97.6	90.0	63.4
Ω	(°)	96.3	313.3	270.0	270.0
ω	(°)	116.7	N/A	N/A	270.0
Segmento Solo		UFMG	UFMG	Polo Norte	Circulo Ártico

4.1.3 Segmento solo

Para cada órbita selecionada, um segmento solo também foi selecionado. Para as órbitas de missões típicas, a Escola de Engenharia da UFMG foi escolhida por ter uma latitude não nula, porém também não extrema (ainda dentro dos trópicos). Já para a órbita polar, o Polo Norte foi selecionado justamente por sua latitude extrema. Por fim, para a órbita Molnya, o Circulo Ártico foi selecionado por ser a região de interesse ao se aplicar este tipo de órbita.

O valor de elevação mínima foi baseado em Donovan (2001), e a velocidade máxima de rastreio foi baseada novamente na *Ground Station* da ISIS (2019). Os parâmetros estão resumidos na Tabela 9.

Tabela 9 – Valores arbitrados para os segmentos solo

		UFMG	Polo Norte	Circulo Ártico
Latitude	(°)	−19.9	90.0	66.5
Longitude	(°)	−44.0	0.0	0.0
Elevação mínima	(°)	10.0	10.0	10.0
Velocidade de rastreio	(°/s)	6.0	6.0	6.0

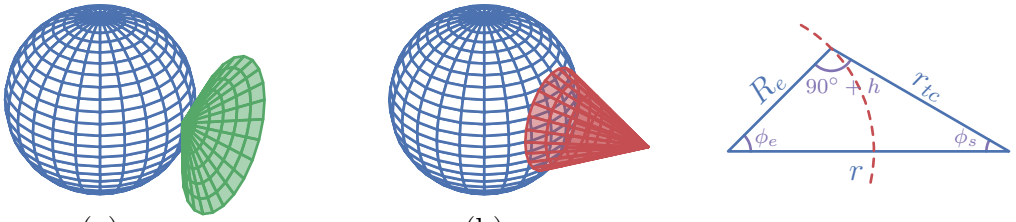
4.2 Variáveis de interesse

Esta seção agrupa as relações analíticas e métodos numéricos obtidos para as variáveis de interesse, conforme descrito no Capítulo 3.

4.2.1 Distância

Assim como é possível definir o campo de visada de uma antena em solo como a região do espaço restrita por um cone de abertura $180° - 2 \cdot h$ centrado na estação solo e com eixo colinear ao centro da Terra, é possível definir a região da Terra com visada para um satélite como o conjunto de pontos contidos em um cone com incidência na superfície terrestre de $90° - h$ centrado no satélite e com eixo colinear ao centro da Terra. A Figura 11 ilustra estas regiões.

Figura 11 – Regiões (a) sob visada do segmento solo e (b) com visada para o segmento voo



(a) (b)

Com isso, aplicando a Lei dos Cossenos no triângulo da Figura 11, obtém-se:

$$r^2 = R_e^2 + r_{tc}^2 - 2 r_{tc} R_e \cos(90° + h) \tag{4.6}$$

$$r_{tc}^2 + 2 R_e \sin(h) r_{tc} + R_e^2 - r^2 = 0 \tag{4.7}$$

$$r_{tc} = \frac{-2 R_e \sin(h) \pm \sqrt{4 R_e^2 \sin^2(h) - 4(R_e^2 - r^2)}}{2} \tag{4.8}$$

Por inspeção, como $r > R_e$ (uma vez que o corpo está em órbita), e $0° \leq h \leq 90°$, tanto a raiz quanto o seno são sempre positivos. Descartando-se a raiz negativa, tem-se:

$$r_{tc}(r, h) = -R_e \sin(h) + \sqrt{r^2 - R_e^2 \cos^2(h)} \tag{4.9}$$

4.2.2 Ângulo de visada

Retornando à Figura 11, nota-se que todas as distâncias estão definidas, porém apenas um ângulo é conhecido. Apesar do triângulo estar totalmente definido, ainda é interessante explicitar os valores para os demais ângulos, uma vez que estes ainda são bastante úteis.

Por exemplo, é mais conveniente obter o ângulo entre os vetores posição dos segmentos voo e solo nos sistema referencial geocêntrico inercial (ϕ_e) do que no sistema topocêntrico (h). Da mesma forma, para se obter o ganho da antena transmissora, é

necessário obter o ângulo de visada dela para o segmento solo. Este ângulo de visada corresponde a ϕ_s quando o sistema de controle de atitude mantém o satélite alinhado ao *nadir*[1], com uma rotação completa por período orbital no plano da órbita.

Ambos os ângulos são obtidos aplicando-se novamente a Lei dos Cossenos:

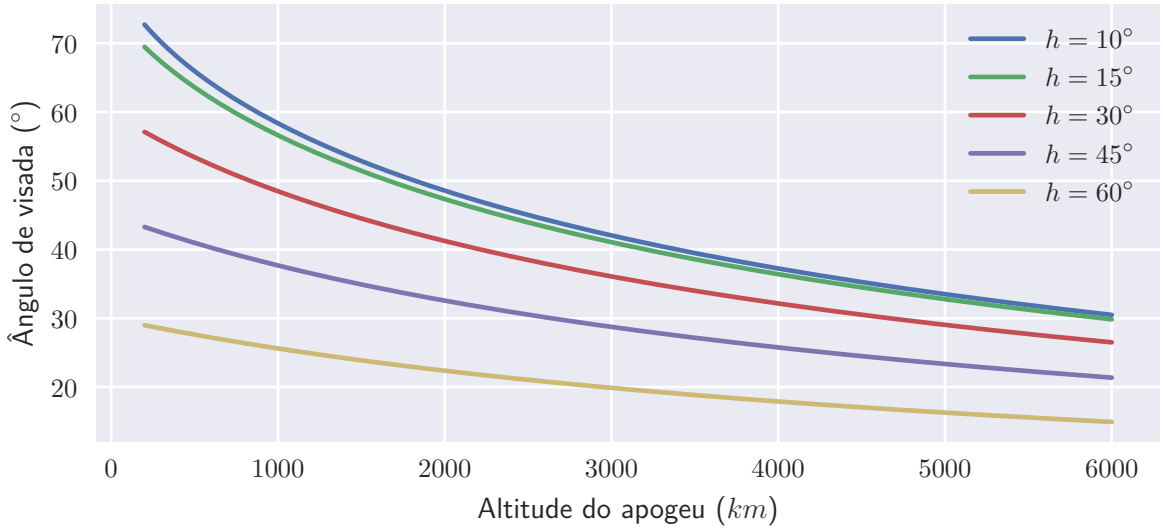
$$r_{tc}(r,h)^2 = R_e^2 + r^2 - 2R_e r\cos(\phi_e) \tag{4.10}$$

$$\phi_e(r,h) = \cos^{-1}\left(\frac{R_e^2 + r^2 - r_{tc}(r,h)^2}{2R_e r}\right) \tag{4.11}$$

$$\phi_s(r,h) = 90^\circ - \phi_e(r,h) - h \tag{4.12}$$

A Figura 12 apresenta a influência da altitude e da elevação no ângulo de visada. Percebe-se que, quanto menor a elevação, maior o ângulo de visada, e portanto menor o ganho em uma antena direcional, porém este efeito se torna menos significativo com o aumento da altitude.

Figura 12 – Ângulo de visada por altitude do apogeu



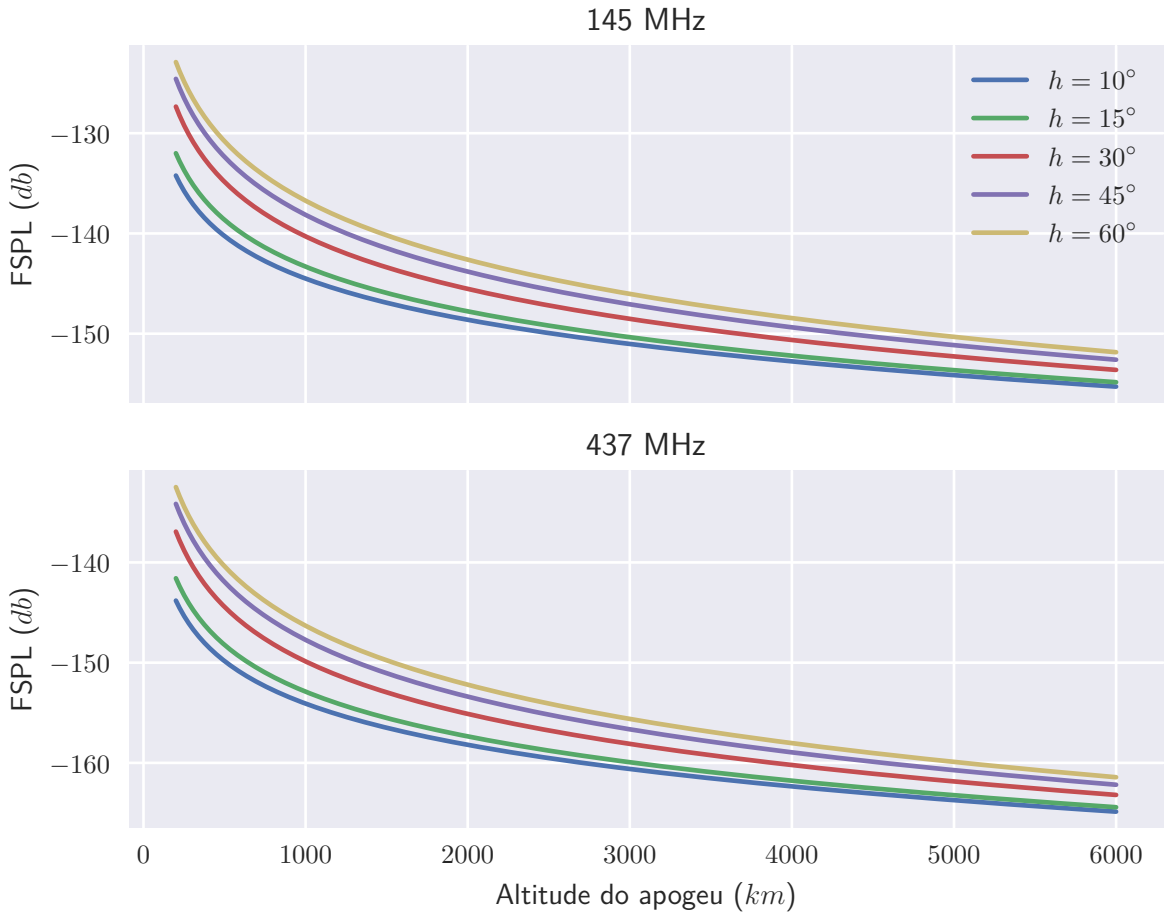
4.2.3 Potência mínima

Para se avaliar a potência mínima requerida para uma missão, é necessário primeiro avaliar as perdas esperadas no sistema. Para uma dada elevação mínima, altitude do apogeu e frequência de interesse, o fator de perda do espaço livre para a órbita pode ser obtido a partir da Equação 2.14, utilizando a Equação 4.9 para se obter a distância máxima entre os segmentos voo e solo.

[1] *Nadir* em um ponto no espaço corresponde à direção da resultante gravitacional no ponto (direção vertical oposta ao zênite)

A Figura 13 apresenta o fator de perda do espaço livre em órbitas baixas para a frequência central das principais bandas levantadas na Seção 2.3.1. Note que este valor varia ao longo de uma janela de comunicação a medida que o ângulo de elevação varia.

Figura 13 – Fator de perda do espaço livre por altitude do apogeu

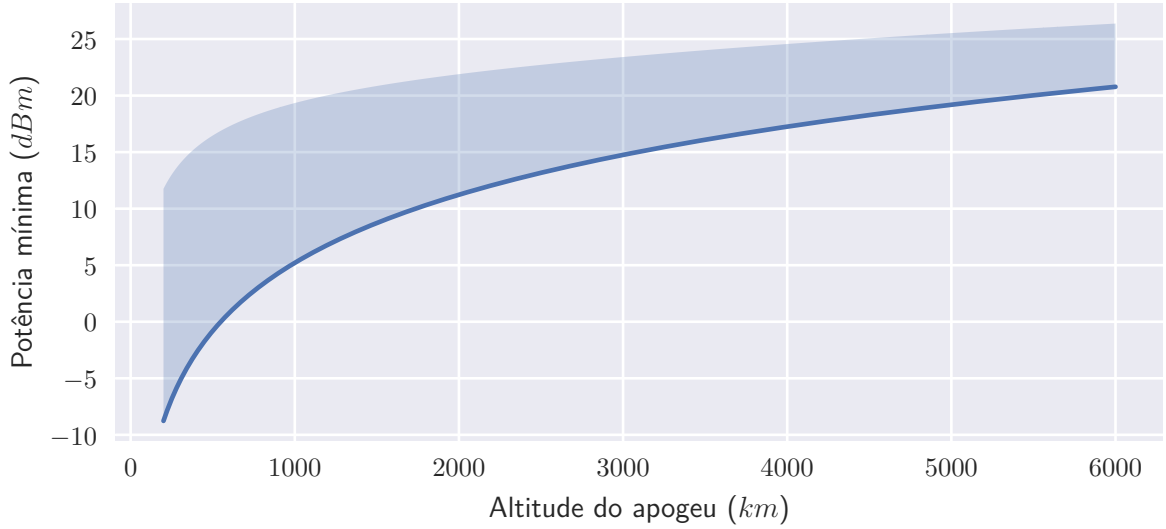


Para um par de antenas isotrópicas com casamento de impedâncias e polarizações compatíveis, o fator de perda do espaço livre corresponde à única atenuação do sistema, e a potência de transmissão mínima pode ser obtida como função direta da sensibilidade do receptor. Já para sistemas reais, os ganhos e eventuais perdas esperadas para o sistema devem ser consideradas, e a potência de transmissão mínima pode ser obtida por meio da Equação 2.15.

A Figura 14 apresenta a potência de transmissão mínima por altitude do apogeu considerando a condição ótima, com distância mínima ($h = 90°$), antenas perfeitamente alinhadas, casamento de impedâncias e polarização compatível. Já a área sombreada representa o aumento na potência mínima para o pior caso[2], com distância máxima (elevação mínima), descasamento de impedâncias e ganho reduzido considerando o ângulo de visada obtido com a Equação 4.12.

[2] Desconsiderando perdas causadas por atenuação atmosférica e erros nos apontamentos.

Figura 14 – Potência de transmissão mínima por altitude do apogeu



4.2.4 Janela de comunicação

A estratégia adotada para se obter a duração das janelas de comunicação consiste em simular as posições do segmento voo e segmento solo, verificando os ângulos entre os vetores posição. A posição do segmento solo é obtida pelas Equações 2.20 e 2.21, o tempo é obtido a partir da Equação 2.18 por meio da anomalia excêntrica, e a posição do segmento solo, considerando uma revolução por dia sideral[3] e a Terra como uma esfera perfeita, é dada por:

$$\vec{r}_{gs} = R_e \cdot \begin{bmatrix} \cos(lon + 360° \cdot t/T_e)\sin(lat) \\ \sin(lon + 360° \cdot t/T_e)\sin(lat) \\ \cos(lat) \end{bmatrix} \tag{4.13}$$

Como $\vec{r}_{gs}$ é colinear ao zênite do sistema topocêntrico, a elevação é dada por:

$$h = 90° - \cos^{-1}\left(\frac{\vec{r}_{gs} \cdot (\vec{r} - \vec{r}_{gs})}{|\vec{r}_{gs}||\vec{r} - \vec{r}_{gs}|}\right) \tag{4.14}$$

4.2.4.1 Órbitas circulares equatoriais

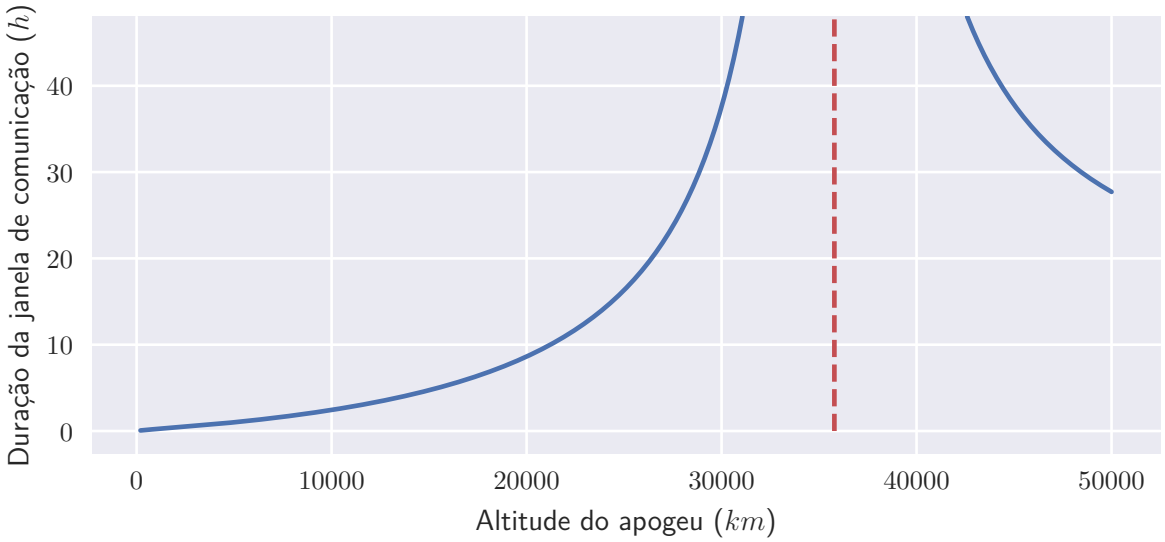
No caso de órbitas circulares equatoriais, toda janela de comunicação é idêntica, e, como as velocidades angulares dos segmentos solo e voo são constantes e estão alinhadas, sua duração apresenta solução analítica:

$$\Delta t = 2 \cdot \frac{\phi_e(|\vec{r}|, h)}{\left|\sqrt{\frac{\mu}{a^3}} - \frac{2\pi}{T_e}\right|} \tag{4.15}$$

[3] Convenciona-se que o meridiano de Greenwich está alinhado ao equinócio vernal em $t = 0$, e portanto o parâmero t_0 da Equação 2.18 é calculado para que o tempo calculado no ponto inicial também seja 0.

A Figura 15 apresenta o a duração da janela de comunicação em função da altitude em órbitas circulares e equatoriais, no qual é possível observar a região das órbitas geossíncronas, em que a duração da janela tende ao infinito (assíntota tracejada), uma vez que o período se aproxima de um dia sideral, e os segmentos voo e solo permanecem sempre alinhados.

Figura 15 – Duração da janela de comunicação em órbitas circulares equatoriais



4.2.4.2 Influência dos parâmetros orbitais na janela de comunicação

No caso de órbitas inclinadas ou excêntricas, a duração da janela de comunicação passa a não ser constante. Para órbitas inclinadas, a janela de comunicação máxima ocorre quando o segmento solo e voo estão perfeitamente alinhados[4], e para órbitas excêntricas, quando este alinhamento ocorre no apogeu da órbita.

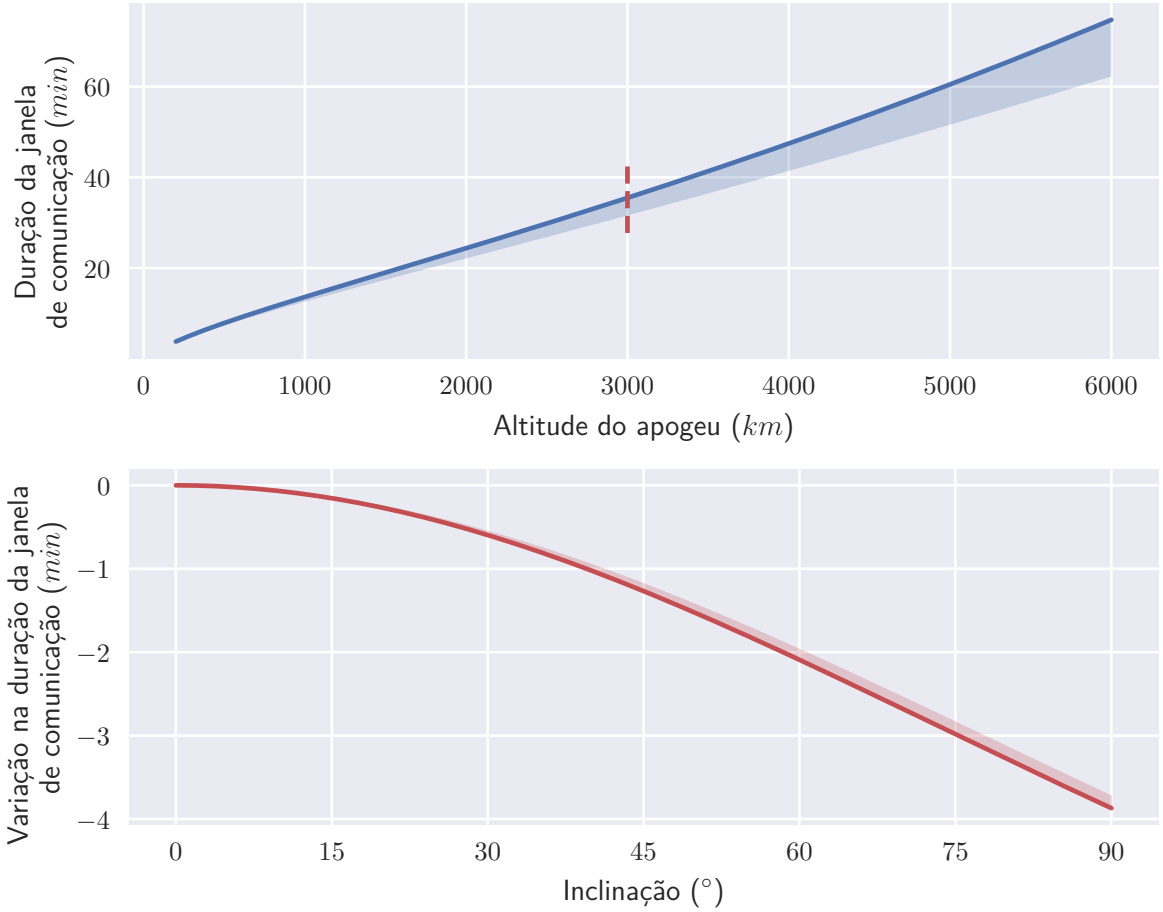
A primeira parte da Figura 16 apresenta a variação máxima na duração da janela de comunicação devido à inclinação da órbita e latitude do segmento solo. A segunda parte da figura apresenta a variação em função da inclinação. A linha sólida corresponde a um segmento solo equatorial, e a área sombreada corresponde a um segmento solo na latitude máxima para a inclinação, ambas para a altitude de referência de 3000 km.

A primeira parte da Figura 17 apresenta a variação máxima na duração da janela de comunicação devido a uma excentricidade de 0.05. A segunda parte da figura apresenta a variação em função da excentricidade para a altitude de referência de 3000 km. Em ambas as partes, o valor mínimo corresponde a uma passagem pelo perigeu, e o valor máximo pelo apogeu.

Para órbitas mais complexas, é recomendada uma análise estatística das durações das janelas de comunicação. A metodologia aqui descrita não considera perturbações na

[4] Apêndice A.1.

Figura 16 – Influência da inclinação e latitude na duração da janela de comunicação



órbita, e é suficiente para um projeto preliminar. Para estágios mais avançados do projeto, o uso de ferramentas como o GMAT (NASA, 2020a), que considera perturbações como arrasto, não esfericidade da Terra, e efeitos gravitacionais de outros corpos celestes, podem ser úteis.

No caso de órbitas pouco excêntricas, é razoável assumir que a distribuição das janelas de comunicação é análoga à distribuição das cordas de um círculo[5]. Nesta distribuição, 90 % dos valores apresentam janelas com durações superiores a 43.6 % do valor máximo, e 50 % apresentam durações superiores a 86.6 %.

$$f(\Delta t) = \frac{\Delta t}{\Delta t_{max}^2 \sqrt{1 - \left(\frac{\Delta t}{\Delta t_{max}}\right)^2}} \tag{4.16}$$

A Figura 18 apresenta um histograma obtido ao simular 2 anos de janelas de comunicação usando a órbita da ISS como referência. Nela também está disposta a distribuição de probabilidade da Equação 4.16.

[5] Apêndice A.2.

Figura 17 – Influência da excentricidade na duração da janela de comunicação

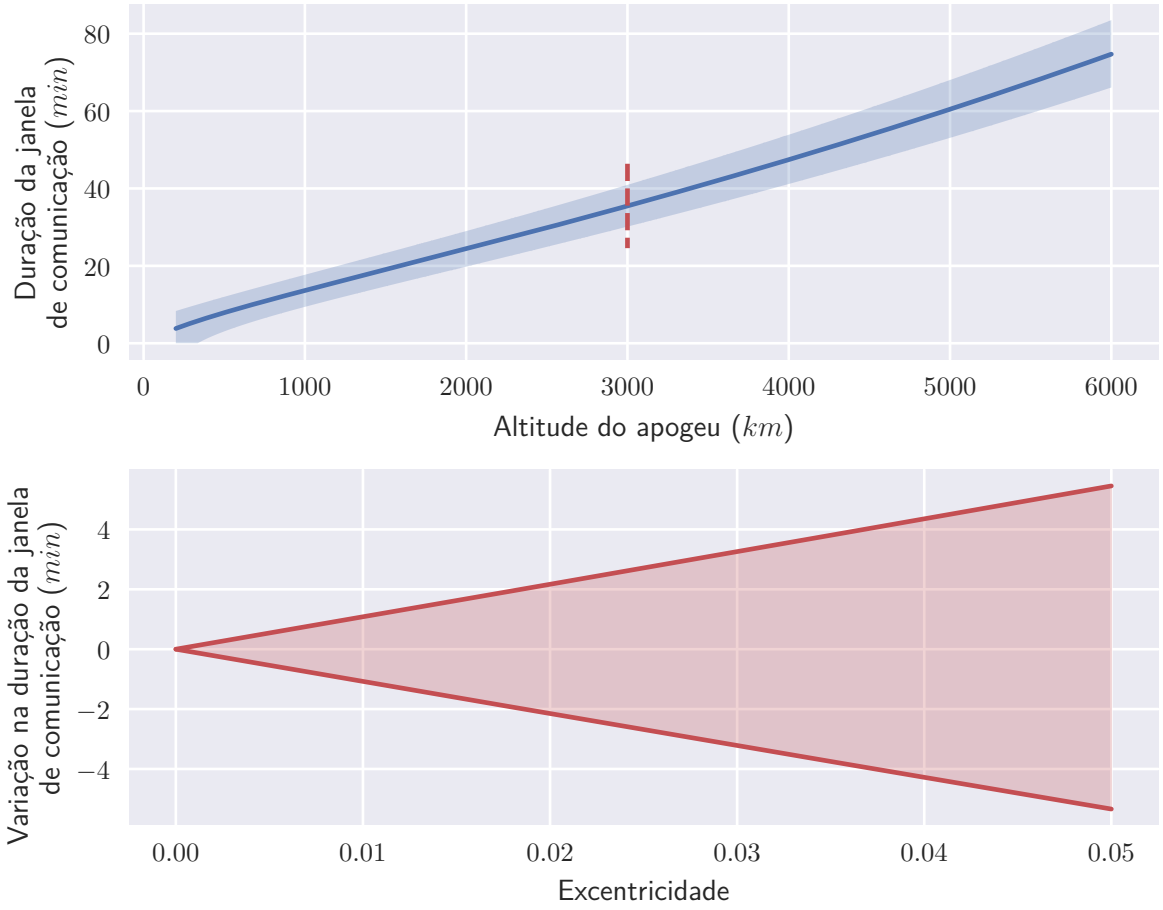
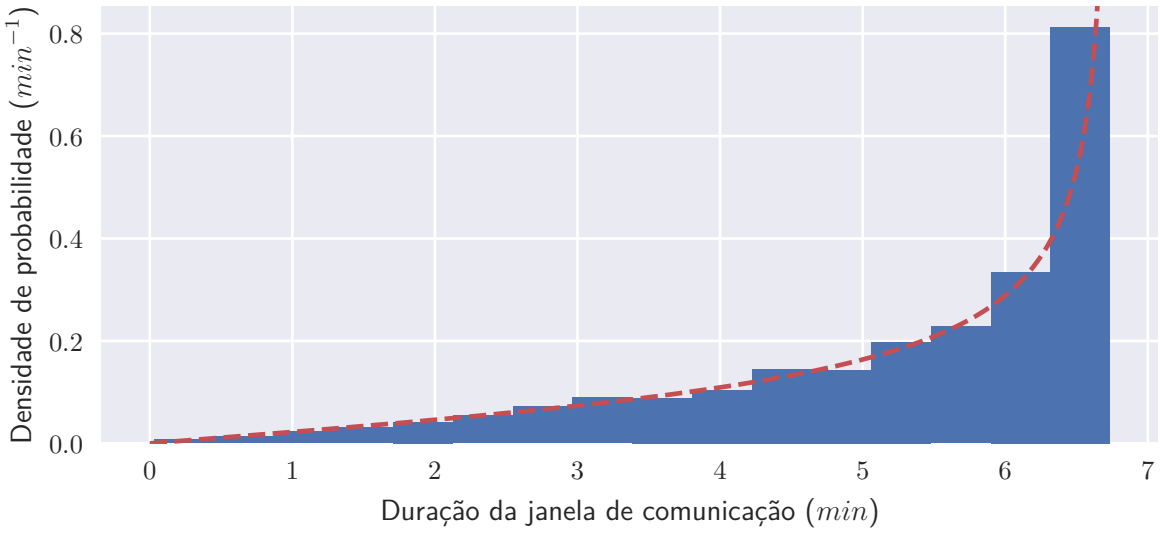


Figura 18 – Distribuição típica da duração da janela de comunicação



4.2.5 Payload transmitida

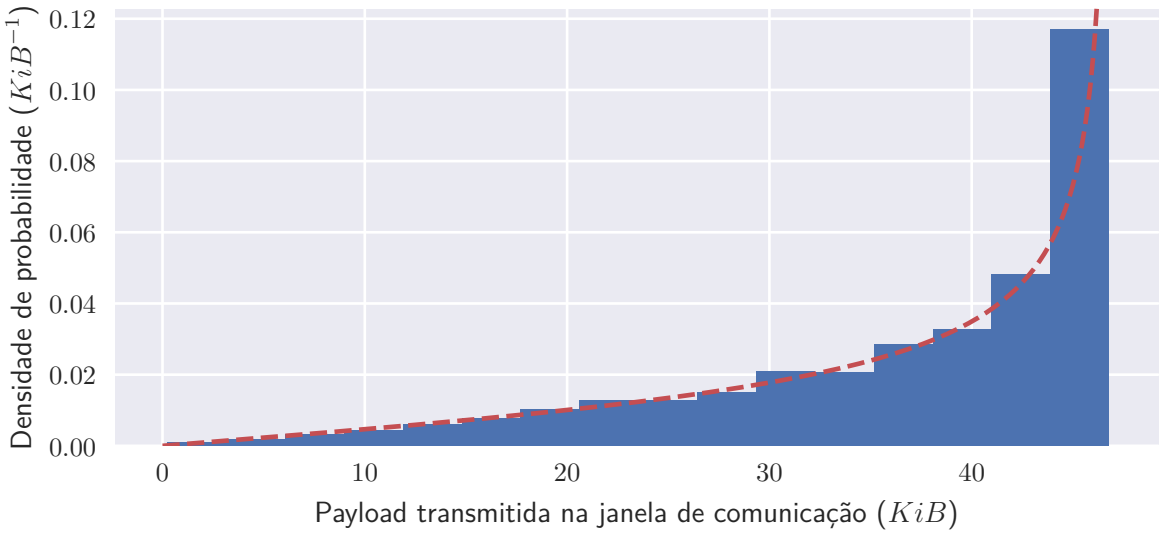
Da Figura 4 e Equação 2.17, considerando o máximo de *frames* consecutivos permitidos, todos com a quantidade máxima de dados por *frame* permitida, tem-se que o tempo para um ciclo de transmissão é:

$$\Delta t = \frac{256}{p+1}\frac{T_{102}}{2} + 2T_{103} + 7 \cdot \frac{63}{62} \cdot \frac{8 \cdot (20+256)}{bitrate} + T_2 + \frac{63}{62} \cdot \frac{8 \cdot 20}{bitrate} \tag{4.17}$$

Utilizando valores típicos para os parâmetros ($T_{102} = 100\,\text{ms}$, $T_{103} = 300\,\text{ms}$, $T_2 = 50\,\text{ms}$ e $p = 63$) e o *bitrate* arbitrado na Seção 4.1, obtém-se um ciclo de transmissão de 15.1 s, que equivale a um *bitrate* efetivo de 948.7 bps.

Este *bitrate* efetivo pode então ser usado para transformar durações da janela de comunicação em tamanho[6] da payload transmitida, como é o caso da Figura 19. É importante destacar que, como a transmissão é realizada em ciclos discretos, é necessário truncar o tamanho da *payload* transmitida na janela de comunicação para um múltiplo inteiro da *payload* transmitida por ciclo.

Figura 19 – Distribuição típica da payload transmitida na janela de comunicação

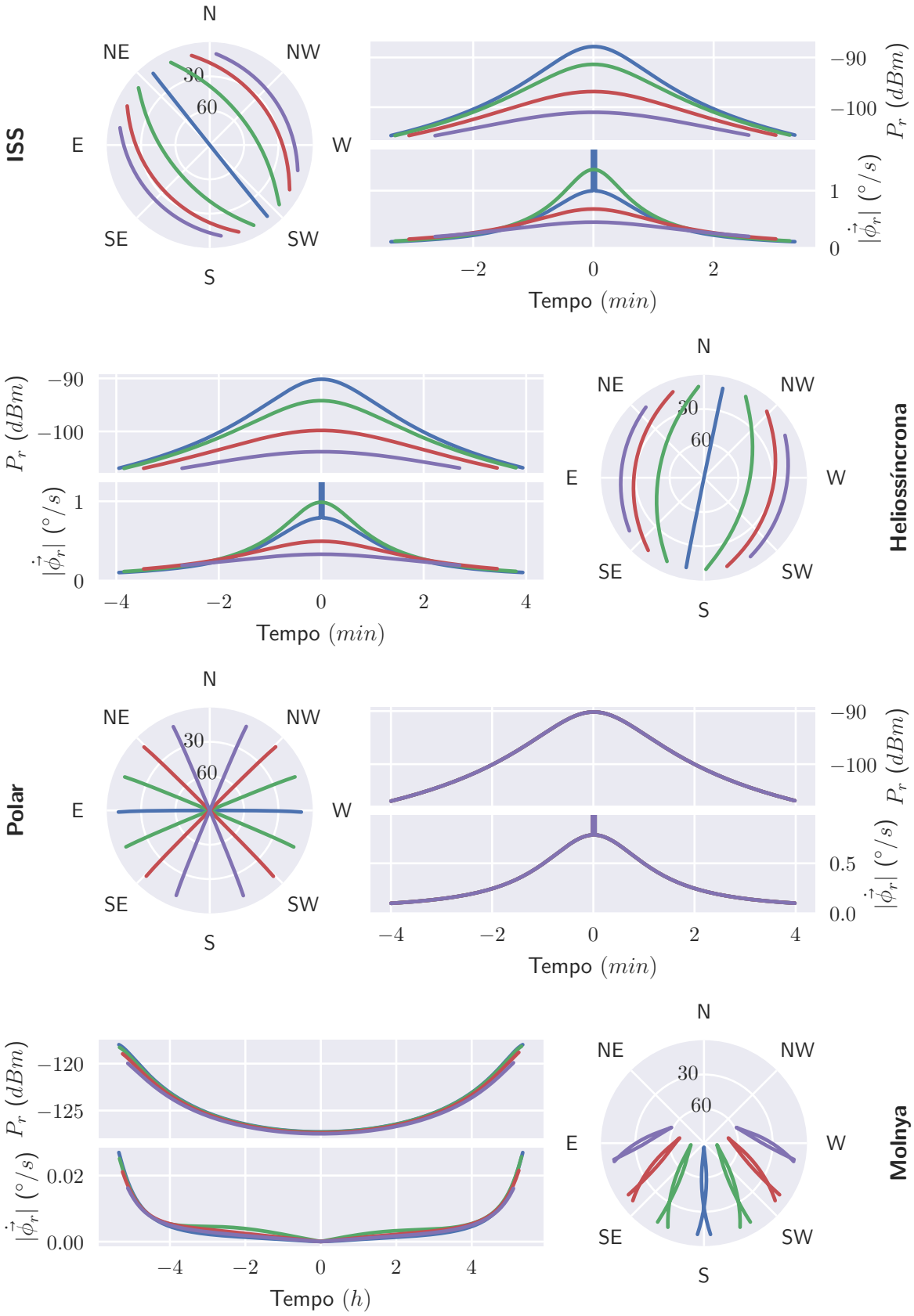


4.3 Simulação em órbitas relevantes

Para cada conjunto de parâmetros orbitais arbitrados na Seção 4.1.2, simularam-se a janelas de comunicação máxima e mínima com desvios angulares igualmente espaçados. Os resultados estão dispostos na Figura 20.

[6] 1 byte (B) = 8 bits (b)
1 kibibyte (KiB) = 1024 bytes (B)

Figura 20 – Simulação das janelas de comunicação de órbitas relevantes



Para a simulação, considerou-se antenas receptoras com movimentação ilimitada no azimute, porém limitada entre 0° e 90° em elevação, o que causa as descontinuidades observadas nas janelas de comunicação máximas.

4.4 Análise de sensibilidade

Esta seção busca averiguar os efeitos das precisões de apontamento e rastreio, que até o momento foram desconsideradas, na potência recebida pelo segmento solo.

4.4.1 Apontamento

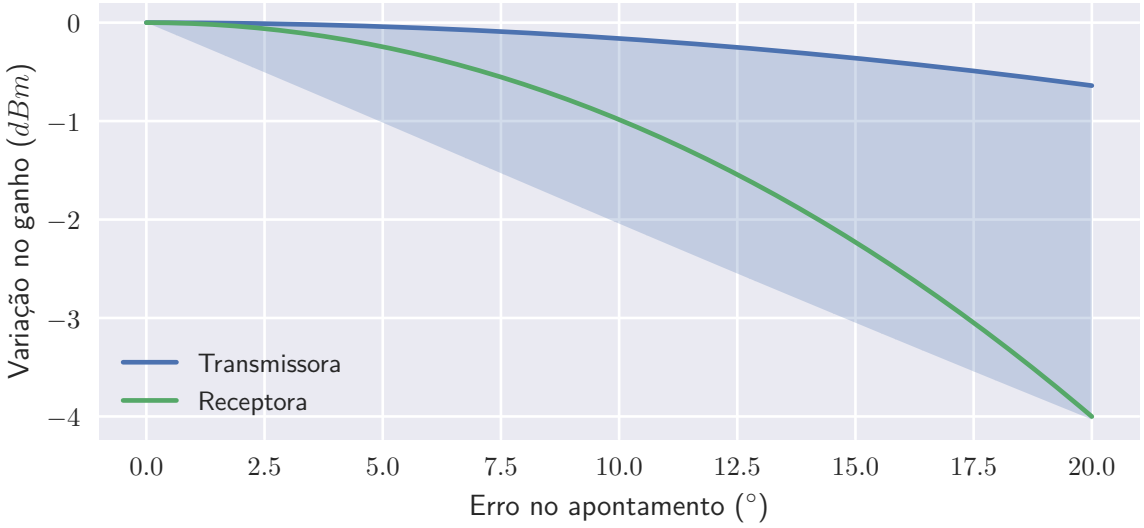
No caso da antena do segmento solo, a referência é a direção do segmento voo, desta forma, a variação do ganho pode ser obtida diretamente em função do erro:

$$\Delta G_{r,dB}(\Delta\theta, \phi) = G_{r,dB}(\Delta\theta, \phi) - G_{0,r,dB} \tag{4.18}$$

Já para a antena do segmento voo, a referência é o *nadir*, e, como visto na Seção 4.2.2, o ângulo de visada varia entre 0° e 90° em função da altitude e elevação.

A Figura 21 apresenta as variações no ganho em função do erro no apontamento para ambas antenas. No caso da antena transmissora, a linha sólida representa a variação no ganho em relação ao ganho máximo, enquanto a área sombreada representa a variação considerando as demais direções como referência.

Figura 21 – Influência do erro de apontamento no ganho



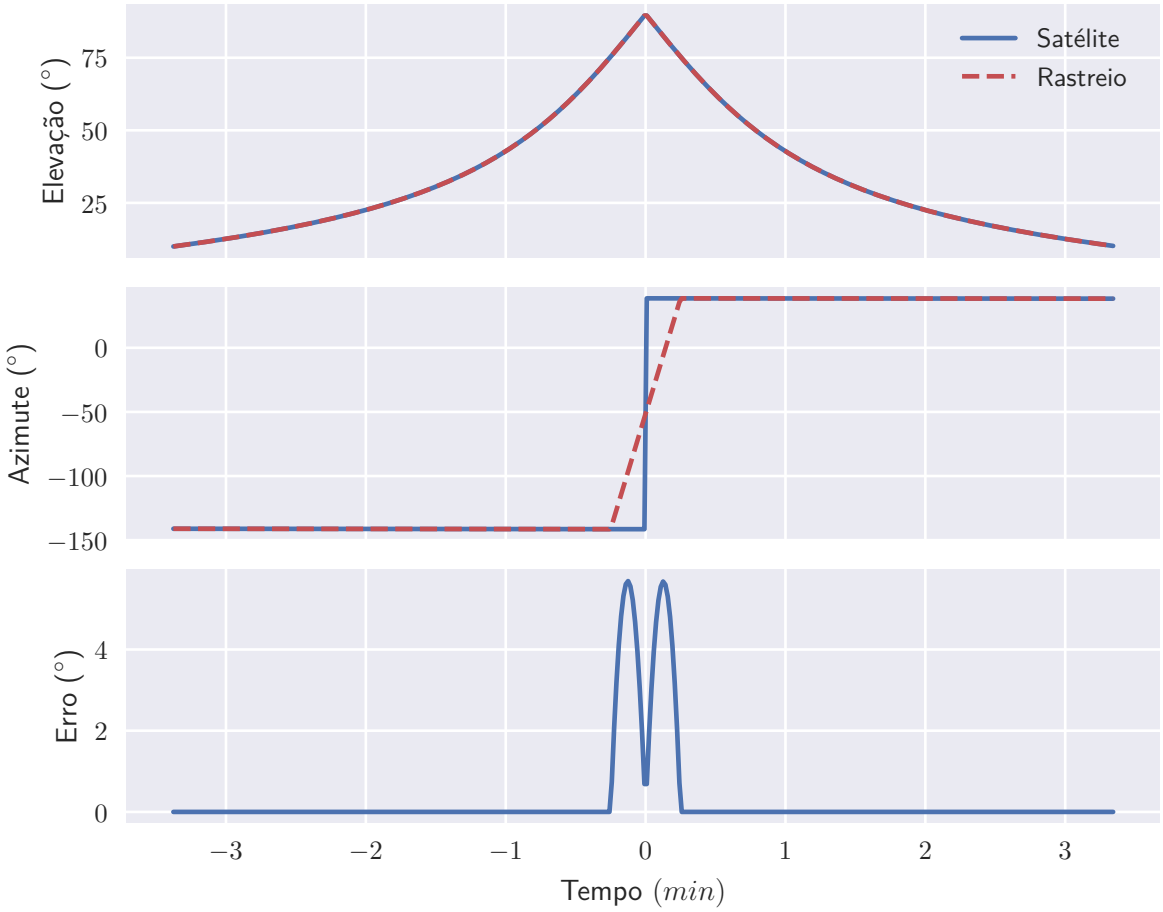
4.4.2 Rastreio

Além da possibilidade de um erro sistemático no alinhamento da antena receptora, é também possível que esta não consiga rastrear corretamente o segmento voo ao longo

da janela, principalmente nos casos que passam pelo ou próximo do zênite, o que causa descontinuidades no rastreio em termos de azimute.

Desta forma, a Figura 22 busca investigar o erro de apontamento gerado ao se limitar a velocidade de rastreio a valores fisicamente possíveis, substituindo descontinuidades por rampas com inclinação máxima definida pela Seção 4.1. Nota-se que, neste caso, o erro de apontamento, assim como a perda de ganho associada são ambos pequenos, além de ocorrerem na região da janela de comunicação em que a potência recebida é máxima.

Figura 22 – Erro de rastreio causado por descontinuidades no azimute



4.5 Necessidades de baixo nível

Finalmente, elaboraram-se necessidades de baixo nível baseadas nas relações analíticas e métodos numéricos obtidos:

1. O sistema de TT&C deve apresentar potência superior à mínima para o apogeu da órbita (Seção 4.2.3), considerando a margem definida em projeto;

2. O sistema de TT&C deve apresentar um *bitrate* efetivo (Seção 4.2.5) suficiente para transmitir os dados de *payload* em uma janela de comunicação típica (Seção 4.2.4);

3. O sistema de TT&C deve apresentar massa igual ou inferior à sua alocação do *budget* de massa;

4. O sistema de controle de atitude deve ser capaz de manter a antena do sistema de TT&C orientada em direção ao *nadir* (premissa da Seção 4.2.2);

5. O sistema de controle de atitude deve apresentar precisão suficiente para que a perda de ganho devido ao erro de apontamento (Seção 4.4.1) seja inferior à margem definida em projeto;

6. O segmento solo deve apresentar precisão suficiente para que a perda de ganho devido ao erro de apontamento (Seção 4.4.1) seja inferior à margem definida em projeto;

7. O segmento solo deve ser capaz de rastrear o segmento voo com velocidade suficiente para que a perda de ganho devido ao erro de apontamento (Seção 4.4.1) seja inferior à margem definida em projeto;

8. O sistema de EPS deve ser capaz de fornecer aos componentes do sistema de TT&C suas respectivas potências de pico;

9. O sistema de TCS deve ser capaz de manter os componentes do sistema de TT&C dentro de suas respectivas faixas operacionais de temperatura.

É importante destacar que os valores obtidos por meio da Seção 4.2 são mínimos para o funcionamento do sistema, e cabe ao projeto definir margens acima dos quais a missão deve operar. Além disso, estas necessidades não levam em conta decisões da arquitetura da missão.

5 Conclusão

Como descrito no Capítulo 1, a justificativa para o desenvolvimento deste trabalho é possibilitar que alunos das graduações em Engenharia Aeroespacial e Engenharia Elétrica sejam capazes de projetar um sistema de TT&C robusto. Com isso, este trabalho abordou os principais conceitos necessários considerando ambas engenharias como público alvo, a metodologia para o levantamento das variáveis de interesse e sistemas relacionados, e o desenvolvimento das relações analíticas e métodos numéricos que embasam o conjunto de necessidades obtido, de forma a possibilitar que demais alunos reproduzam os passos aqui descritos para projetar um sistema de TT&C a nível de graduação.

A principal sugestão para trabalhos futuros é a aplicação do aprendizado deste trabalho no projeto de um *CubeSat*, como por exemplo o do PdQSat-1. Ainda neste contexto, existem vantagens em se desenvolver uma *ground station* sem o uso de componentes COTS, e algumas das considerações deste trabalho podem servir como um ponto inicial, especialmente para o projeto da antena e do sistema de controle de apontamento.

Já no contexto do modelo utilizado, uma possibilidade é integrá-lo com os modelos de outros sistemas para uma simulação mais completa do comportamento do segmento voo. Por fim, existem também oportunidades de melhoria no modelo em sí, dos quais se destacam:

- Considerar atenuações atmosféricas, com modelos de propagação com densidade variável e ambientes chuvosos ou nublados;
- Considerar ruido térmico da antena, e possíveis interferências eletromagnéticas;
- Considerar perturbações orbitais, como a não esfericidade da Terra, e efeitos gravitacionais da Lua e Sol;
- Verificar os ganhos de implementações com protocolos mais modernos, ou bandas de frequências levantadas como promissoras em NASA (2020b).

Referências

ALMEIDA, D. P.; MATTIELLO-FRANCISCO, F. Modeling of the interoperability between on-board computer and payloads of the NanosatC-BR2 with support of the UPPAAL tool. In: *1st IAA Latin American Symposium on Small Satellites*. Buenos Aires: [s.n.], 2017. v. 9. Citado na página 33.

ALVES, A. et al. CONASAT-0: Visão geral do nanossatélite desenvolvido. In: CONGRESSO AEROESPACIAL BRASILEIRO, 2., 16-19 Set., Santa Maria. *Anais...* Santa Maria, 2019. Citado na página 33.

ANATEL. Plano de atribuição, destinação e distribuição de frequências no Brasil. 2020. Citado na página 31.

BALANIS, C. A. *Antenna theory: analysis and design*. 4. ed. Hoboken, New Jersey: John Wiley & Sons, 2016. Citado na página 19.

BEECH, W. A.; NIELSEN, D. E.; NOO, J. T. AX.25 link access protocol for amateur packet radio. *Tucson Amateur Packet Radio Corporation*, n. 2.2, 1998. Citado 3 vezes nas páginas 19, 23 e 24.

BEZERRA, E. A. et al. *FloripaSat TT&C: Telemetry, Tracking and Command Module of the FloripaSat Project*. 2019. Disponível em: <https://github.com/floripasat/ttc>. Acesso em: 18 Jul. 2021. Citado 2 vezes nas páginas 33 e 34.

BRAGA, J. et al. LECX: a cubesat experiment to detect and localize cosmic explosions in hard x-rays. *Monthly Notices of the Royal Astronomical Society*, Oxford University Press, v. 493, n. 4, p. 4852–4860, 2020. Citado na página 33.

BRUDER, J. et al. IEEE standard for letter designations for radar-frequency bands. *IEEE Aerospace & Electronic Systems Society*, n. 521, 2019. Citado na página 19.

CHOBOTOV, V. A. *Orbital mechanics*. 3. ed. Reston, Virginia: AIAA, 2002. Citado na página 25.

COSTA, L. Z. da et al. Programa NanosatC-BR: Resultados obtidos e perspectivas futuras. Oct 2017. Citado na página 33.

CURTIS, H. *Orbital mechanics for engineering students*. Oxford: Butterworth-Heinemann, 2005. Citado na página 25.

DONOVAN, H. *Reduction of the Minimum Elevation Angle for NASA Satellite Laser Ranging Tracking Operations*. 2001. Disponível em: <https://cddis.nasa.gov/lw12/docs/Donovan_Reduction%20in%20the%20Minimum%20Elevation.pdf>. Acesso em: 25 Dez. 2021. Citado na página 39.

ENDUROSAT. *DATASHEET UHF Antenna*. [S.l.], 2021. Disponível em: <https://satsearch.co/products/endurosat-uhf-antenna>. Acesso em: 10 Nov. 2021. Citado na página 35.

ENDUROSAT. *DATASHEET UHF Transceiver*. [S.l.], 2021. Disponível em: <https://satsearch.co/products/endurosat-uhf-transceiver-ii>. Acesso em: 10 Nov. 2021. Citado na página 34.

INPE. *Diário de bordo: Setembro 2016 - Julho 2017*. 2017. Disponível em: <http://www.crn.inpe.br/conasat1/situ_setjul_2017.php>. Acesso em: 18 Jul. 2021. Citado na página 33.

ISIS. *Antenna Systems*. [S.l.], 2016. Disponível em: <https://www.cubesatshop.com/wp-content/uploads/2016/06/ISIS-Antenna-systems-Brochure-v1.pdf>. Acesso em: 10 Nov. 2021. Citado na página 35.

ISIS. *VHF/UHF duplex transceiver*. [S.l.], 2016. Disponível em: <https://www.cubesatshop.com/wp-content/uploads/2016/06/VHF-UHF-Full-Duplex-Transceiver-Brochure-web-1.pdf>. Acesso em: 10 Nov. 2021. Citado na página 34.

ISIS. *VHF/UHF Ground Station Kit Data Sheet*. [S.l.], 2019. Disponível em: <https://www.isispace.nl/wp-content/uploads/2016/02/ISIS-GSKit-DS-302_v1.2-VHFUHF-Ground-Station-DataSheet-for-website.pdf>. Acesso em: 15 Nov. 2021. Citado 2 vezes nas páginas 37 e 39.

ISIS. *Deployable Antenna System*. [S.l.], 2020. Disponível em: <https://www.isispace.nl/wp-content/uploads/2021/01/ISIS-ANTS-DSH-0001-Antenna_System_Datasheet-04_00.pdf>. Acesso em: 10 Nov. 2021. Citado na página 35.

ISIS. *CubeSatShop.com*. 2021. Disponível em: <https://www.cubesatshop.com/>. Acesso em: 18 Jul. 2021. Citado 2 vezes nas páginas 31 e 32.

ITU. Nomenclature of the frequency and wavelength bands used in telecommunications. *Recommendation ITU/RV*, n. V.431-8, 2015. Citado na página 19.

JOHNSTONE, A. et al. Cubesat design specification. 2020. Citado na página 16.

KLESH, A. et al. MarCO: Early operations of the first CubeSats to Mars. 2018. Citado na página 16.

KUGA, H. K.; KONDAPALLI, R. R.; CARRARA, V. *Introdução à mecânica orbital*. 2. ed. Sao José dos Campos: INPE, 2012. 67 p. Disponível em: <http://urlib.net/rep/8JMKD3MGPAW/3C76K98>. Acesso em: 25 Dez. 2021. Citado 2 vezes nas páginas 25 e 27.

KULU, E. Nanosats database. Apr 2021. Disponível em: <https://www.nanosats.eu/>. Acesso em: 18 Jul. 2021. Citado 3 vezes nas páginas 16, 29 e 30.

MEHRPARVAR, A. et al. Cubesat design specification. *The CubeSat Program*, San Luis Obispo, 2014. Citado na página 16.

NASA. *General Mission Analysis Tool*. 2020. Disponível em: <https://sourceforge.net/projects/gmat>. Acesso em: 29 Dez. 2021. Citado na página 45.

NASA. *State of the Art of Small Spacecraft Technology*. 2020. Disponível em: <https://www.nasa.gov/smallsat-institute/sst-soa-2020>. Acesso em: 18 Jul. 2021. Citado 5 vezes nas páginas 16, 19, 31, 32 e 52.

NASA. *SmallSat Parts On Orbit Now*. 2021. Disponível em: <https://spoonsite.com/>. Acesso em: 18 Jul. 2021. Citado 2 vezes nas páginas 31 e 32.

NICERF. *VHF/UHF duplex transceiver*. [S.l.], 2018. Disponível em: <https://www.nicerf.com/9f66ac9f-b452-413a-966a-2a9dacbf047c>. Acesso em: 10 Nov. 2021. Citado na página 34.

SATO, L. H. S. et al. The ITASAT: The lessons learned from the mission concept to the operation. 2019. Citado na página 33.

SCHUCH, N. J. et al. Gestão de competências e futuro do programa NanosatC-BR, desenvolvimento de CubeSats, parceria UFSM - INPE/MCTIC. In: CONGRESSO AEROESPACIAL BRASILEIRO, 2., 16-19 Set., Santa Maria. *Anais...* Santa Maria, 2019. Disponível em: <http://dx.doi.org/10.29327/2cab2019.224956>. Acesso em: 27 Set. 2021. Citado na página 33.

ZIELINSKI, B. M. Effective transmission speed in AX.25 protocol. In: *IEEE EUROCON 2009*. [S.l.: s.n.], 2009. p. 1763–1768. Citado 3 vezes nas páginas 19, 24 e 25.

Apêndices

APÊNDICE A – Deduções

A.1 Algoritmo de alinhamento

Considere o sistema de coordenadas rotacionado em torno de Z tal que, no instante de referência t_0, os vetores posição dos segmentos solo e voo são colineares entre si, e coplanares ao novo plano XZ.

Desta forma, a latitude do segmento solo se mantém, porém sua nova longitude é $lon = 0$. Da mesma forma, dentre os elementos Keplerianos, apenas a ascensão reta do nodo ascendente é afetada. Como a inclinação corresponde ao ângulo entre o vetor $\hat{k}$ e o vetor normal ao plano da órbita ($\hat{k}_{pf}$), sua projeção em Z é:

$$\hat{k}_{pf} \cdot \hat{k} = \cos(i) \Rightarrow \hat{k}_{pf,z} = \cos(i) \tag{A.1}$$

A projeção de $\hat{k}_{pf}$ em X é obtida de sua perpendicularidade com $\hat{r}$:

$$\hat{k}_{pf} \perp \hat{r} \Leftrightarrow \hat{k}_{pf} \cdot \hat{r} = 0 \Rightarrow \hat{k}_{pf,x} \cdot \cos(lat) + \hat{k}_{pf,y} \cdot 0 + \cos(i) \cdot \sin(lat) \Rightarrow \hat{k}_{pf,x} = -\cos(i)\tan(lat) \tag{A.2}$$

Por fim, como $\hat{k}_{pf}$ é unitário, sua projeção em Y é:

$$|\hat{k}_{pf}| = 1 \Rightarrow \sqrt{\cos(i)^2 \tan(lat)^2 + \hat{k}_{pf,y}^2 + \cos(i)^2} = 1 \Rightarrow \hat{k}_{pf,y} = \pm\sqrt{1 - \cos^2(i)(1 + \tan^2(lat))} \tag{A.3}$$

Em suma:

$$\hat{k}_{pf} = \begin{bmatrix} -\cos(i)\tan(lat) \\ \pm\sqrt{1 - \cos^2(i)(1 + \tan^2(lat))} \\ \cos(i) \end{bmatrix} \tag{A.4}$$

A ascensão reta do nodo ascendente é então:

$$\vec{\Omega} = \hat{k} \times \hat{k}_{pf} \tag{A.5}$$

$$\Omega = \tan^{-1}\left(\frac{\vec{\Omega}_y}{\vec{\Omega}_x}\right) \tag{A.6}$$

E a anomalia verdadeira é obtida transformando $\hat{r}$ para o sistema perifocal:

$$\hat{r}_{pf} = R(\Omega, i, \omega)\hat{r} \tag{A.7}$$

$$f = \tan^{-1}\left(\frac{\hat{r}_{pf,y}}{\hat{r}_{pf,x}}\right) \tag{A.8}$$

A.2 Distribuição de cordas em um círculo

Dada a função $g : \mathbb{R} \to \mathbb{R}$ com domínio $\{x \in \mathbb{R} | 0 \leq x \leq R\}$ que retorna o tamanho da corda de um círculo de raio R distante em x do centro:

$$g(x) = R\sin(\cos^{-1}(x/R)) \tag{A.9}$$

Como g é monótona, sua inversa é:

$$g^{-1}(y) = R\cos(\sin^{-1}(y/R)) \tag{A.10}$$

Desta forma, a densidade de probabilidade das cordas, dada uma distribuição uniforme $U_{[0,R]}$ nas distâncias, é:

$$f(y) = U_{[0,R]}(g^{-1}(y))\left|\frac{d}{dy}(g^{-1}(y))\right| = \frac{1}{R}\left|\frac{-y}{R\sqrt{1-y^2/R^2}}\right| = \frac{y}{R^2\sqrt{1-y^2/R^2}} \tag{A.11}$$

A densidade cumulativa é:

$$P(0 < y \leq a) = \int_0^a f(y)dy = 1 - \sqrt{1 - \frac{a^2}{R^2}} \tag{A.12}$$

E o quantil é:

$$Q(p) = P^{-1}(p) = R\sqrt{2p - p^2} \tag{A.13}$$

APÊNDICE B – Código fonte

Todo código desenvolvido para este trabalho está disponível em <https://github.com/bss-aero/cubesat-ttc-utils>, e está dividido nos arquivos:

- **constants.py:** Constantes físicas relacionadas ao problema;
- **link_distance.py:** Cálculos de distância e ângulos em função de apogeu e elevação mínima descritos nas Seções 4.2.1 e 4.2.2;
- **contact.py:** Objetos para segmentos solo e voo e antenas, com métodos para simulação de janelas de comunicação;
- **timed_contact.py:** Rotinas para obtenção das durações de janela de comunicação da Seção 4.2.4;
- **ax25_times.py:** Cálculos de tempo de duração do *frame* e *bitrate* efetivo da Seção 4.2.5;
- **plot.py:** Rotina de cálculo das imagens dos gráficos do Capítulo 4

B.1 constants.py

```
SPEED_OF_LIGHT = 299792458  # m/s
BODY_RADIUS = 6378e3  # m
BODY_PERIOD = 86164.09053083288  # s (sidereal day)
BODY_MU = 3.986004418e14  # m^3 s^-2
BODY_J2 = 1.08263e-3  # m^5 s^2

AX25_T102 = 100  # ms
AX25_T103 = 300  # ms
AX25_T2 = 50  # ms
AX25_P = 63
```

B.2 link_distance.py

```
import numpy as np

from constants import *


def angle_from_sides(a, b, c):
    return np.arccos((c ** 2 + b ** 2 - a ** 2) / (2 * b * c))


def get_link_distance(r, elev):
```

```
    return -BODY_RADIUS * np.sin(elev) + np.sqrt(r ** 2 - BODY_RADIUS ** 2 * np.cos(elev)
        ** 2)


def get_link_aperture_body(r, elev):
    link_distance = get_link_distance(r, elev)
    return angle_from_sides(link_distance, BODY_RADIUS, r)


def get_link_aperture_sat(r, elev):
    link_distance = get_link_distance(r, elev)
    return angle_from_sides(BODY_RADIUS, link_distance, r)
```

B.3 contact.py

```
import numpy as np

from constants import *
from link_distance import angle_from_sides

INTEGRATION_RESOLUTION = 1024


def to_db(x):
    return 10 * np.log10(x)


def rot_x(theta):
    c = np.cos(theta)
    s = np.sin(theta)
    return np.array([[1, 0, 0], [0, c, s], [0, -s, c]])


def rot_z(theta):
    c = np.cos(theta)
    s = np.sin(theta)
    return np.array([[c, s, 0], [-s, c, 0], [0, 0, 1]])


def get_angle(v1, v2):
    return np.arccos(np.sum((v1 / np.linalg.norm(v1, axis=0)) * (v2 / np.linalg.norm(v2,
        axis=0)), axis=0))


class Spacecraft(object):
    def __init__(self, sma, ecc, inc, raan, aop, f0):
        self.sma = sma
        self.ecc = ecc
        self.inc = np.radians(inc)
        self.raan = np.radians(raan)
        self.aop = np.radians(aop)
        self.f0 = np.radians(f0)

        self.n = None
        self.u0 = None
        self.t0 = 0
        self.rot = None
        self.period = None
        self.antenna = None
```

```
44          self.update()
45
46      def update(self):
47          self.n = np.sqrt(BODY_MU / self.sma ** 3)
48          self.t0 = 0
49          self.u0 = np.arctan2(np.sqrt(1 - self.ecc ** 2) * np.sin(self.f0), self.ecc + np.
                cos(self.f0))
50          self.t0 = self.get_time(0)
51          self.rot = rot_z(-self.raan).dot(rot_x(-self.inc)).dot(rot_z(-self.aop))
52          self.period = self.get_time(2 * np.pi)
53
54      def set_antenna(self, antenna):
55          self.antenna = antenna
56
57      def get_local_position(self, u):
58          if not isinstance(u, np.ndarray):
59              u = np.array(u)
60
61          return np.array([
62              self.sma * (np.cos(u + self.u0) - self.ecc),
63              self.sma * np.sin(u + self.u0) * (1 - self.ecc ** 2) ** .5,
64              np.zeros(u.shape)
65          ])
66
67      def get_global_position(self, u):
68          return self.rot.dot(self.get_local_position(u))
69
70      def get_time(self, u):
71          if not isinstance(u, np.ndarray):
72              u = np.array(u)
73
74          return (u + self.u0 - self.ecc * np.sin(u + self.u0)) / self.n - self.t0
75
76      def as_copy(self, **kwargs):
77          args = {
78              'sma': self.sma,
79              'ecc': self.ecc,
80              'inc': np.degrees(self.inc),
81              'raan': np.degrees(self.raan),
82              'aop': np.degrees(self.aop),
83              'f0': np.degrees(self.f0)
84          }
85          args.update(kwargs)
86          result = Spacecraft(**args)
87          result.set_antenna(self.antenna)
88          return result
89
90
91  class GroundStation(object):
92      def __init__(self, lat, lon, min_elev):
93          self.lat = np.radians(lat)
94          self.lon = np.radians(lon)
95          self.min_elev = np.radians(min_elev)
96          self.antenna = None
97
98      def set_antenna(self, antenna):
99          self.antenna = antenna
100
101     def get_effective_longitude(self, time):
102         return 2 * np.pi * (time / BODY_PERIOD % 1) + self.lon
```

```
103
104     def get_global_position(self, time):
105         if not isinstance(time, np.ndarray):
106             time = np.array(time)
107
108         lon = self.get_effective_longitude(time)
109         phi = np.ones(lon.shape) * (np.pi / 2 - self.lat)
110         return BODY_RADIUS * np.array([
111             np.cos(lon) * np.sin(phi),
112             np.sin(lon) * np.sin(phi),
113             np.cos(phi)
114         ])
115
116     def has_sight(self, sc, u):
117         time = sc.get_time(u)
118         gs_pos = self.get_global_position(time)
119         sc_pos = sc.get_global_position(u)
120         angles = get_angle(gs_pos, sc_pos - gs_pos)
121
122         return (
123             TimeHistory(self, time, u, gs_pos),
124             TimeHistory(sc, time, u, sc_pos),
125             angles <= np.pi / 2 - self.min_elev
126         )
127
128     def as_copy(self, **kwargs):
129         args = {
130             'lat': np.degrees(self.lat),
131             'lon': np.degrees(self.lon),
132             'min_elev': np.degrees(self.min_elev)
133         }
134         args.update(kwargs)
135         result = GroundStation(**args)
136         result.set_antenna(self.antenna)
137         return result
138
139
140 class TimeHistory(object):
141     def __init__(self, parent, time, u, position):
142         self.parent = parent
143         self.time = time
144         self.u = u
145         self.position = position
146
147     def __len__(self):
148         return len(self.time)
149
150     def __getitem__(self, item):
151         return TimeHistory(self.parent, self.time[item], self.u[item], self.position[:,
                item])
152
153
154 class Contact(object):
155     def __init__(self, gs, sc, partial):
156         self.gs = gs
157         self.sc = sc
158         self.partial = partial
159
160         self.start = sc.time[0]
161         self.end = sc.time[-1]
```

```
162         self.duration = self.end - self.start
163
164         self._elevation = None
165         self._azimuth = None
166         self._tracking = None
167         self._power = None
168
169     def update(self):
170         sc_pos_local = self.sc.position - self.gs.position
171         gs_pos_unit = self.gs.position / np.linalg.norm(self.gs.position, axis=0)
172
173         self._elevation = get_angle(self.gs.position, sc_pos_local)
174         sc_pos_plane = sc_pos_local - np.linalg.norm(sc_pos_local, axis=0) * np.cos(self.
                _elevation) * gs_pos_unit
175
176         east_theta = np.pi / 2 + self.gs.parent.get_effective_longitude(self.gs.time)
177         east = np.array([
178             np.cos(east_theta),
179             np.sin(east_theta),
180             np.zeros(east_theta.shape)
181         ])
182         north = np.cross(gs_pos_unit, east, axis=0)
183
184         azimuth_north = get_angle(north, sc_pos_plane)
185         azimuth_east = get_angle(east, sc_pos_plane)
186
187         self._azimuth = np.where(azimuth_east <= np.pi / 2, azimuth_north, -azimuth_north
                )
188
189         if (self.gs.parent.antenna is not None) and (self.sc.parent.antenna is not None):
190             gs_dist = np.linalg.norm(self.gs.position, axis=0)
191             sc_dist = np.linalg.norm(self.sc.position, axis=0)
192             sc_dist_local = np.linalg.norm(sc_pos_local, axis=0)
193
194             self._power = self.sc.parent.antenna.transmit(
195                 self.gs.parent.antenna,
196                 sc_dist_local,
197                 angle_from_sides(gs_dist, sc_dist, sc_dist_local),
198                 0
199             )
200
201             deltas = 2 * np.ones(self._azimuth.shape)
202             deltas[0] = 1
203             deltas[-1] = 1
204
205             diff_az = deltas * np.gradient(self._azimuth)
206             diff_az_compl = diff_az - np.sign(diff_az) * 2 * np.pi
207             gradient_az = np.where(abs(diff_az) > np.pi, diff_az_compl, diff_az) / (
                    deltas * np.gradient(self.sc.time))
208             gradient_elev = np.gradient(self._elevation, self.sc.time)
209
210             self._tracking = np.stack((gradient_az, gradient_elev))
211
212     @property
213     def elevation(self):
214         if self._elevation is None:
215             self.update()
216         return np.pi / 2 - self._elevation
217
218     @property
```

```
219     def azimuth(self):
220         if self._azimuth is None:
221             self.update()
222         return self._azimuth
223 
224     @property
225     def power(self):
226         if self._power is None:
227             self.update()
228         return self._power
229 
230     @property
231     def tracking(self):
232         if self._tracking is None:
233             self.update()
234         return self._tracking
235 
236     @staticmethod
237     def get_contacts(u, gs, sc, partials=False):
238         th_gs, th_sc, has_sight = gs.has_sight(sc, u)
239         edges = list(np.where(has_sight[:-1] != has_sight[1:])[0])
240 
241         start_edge = bool(has_sight[0])
242         end_edge = bool(has_sight[-1])
243 
244         if start_edge:
245             if partials:
246                 edges.insert(0, 0)
247             else:
248                 edges.pop(0)
249 
250         if partials and end_edge:
251             edges.append(None)
252 
253         contacts = list()
254 
255         for limits in zip(edges[::2], edges[1::2]):
256             sliced = slice(*limits)
257             partial = (limits[0] == 0) or (limits[1] is None)
258             contacts.append(
259                 Contact(th_gs[sliced], th_sc[sliced], partial)
260             )
261 
262         return contacts
263 
264 
265 class Antenna(object):
266     def __init__(self, freq, vswr=1, radiation_efficiency=1, polarization_efficiency=1,
        radiation_pattern=None,
267                  transmit_pwr=None, sensibility=None):
268         assert radiation_efficiency <= 1
269         self.freq = freq
270         self.vswr = vswr
271         self.radiation_efficiency = radiation_efficiency
272         self.reflection_efficiency = (1 - ((vswr - 1) / (vswr + 1)) ** 2)
273         self.polarization_efficiency = polarization_efficiency
274         self.losses = to_db(self.radiation_efficiency * self.reflection_efficiency * self
            .polarization_efficiency)
275         self.transmit_pwr = transmit_pwr
276         self.sensibility = sensibility
```

```
277
278         self._radiation_pattern = radiation_pattern
279         self._directivity = None
280         self._gain = None
281         self.hpbw = None
282         self.p_rad = None
283         self.g0 = 0
284
285         if radiation_pattern is not None:
286             self.update()
287
288     def update(self, res=INTEGRATION_RESOLUTION):
289         elev = np.linspace(0, np.pi, res)
290         self.p_rad = 2 * np.pi * np.trapz(self._radiation_pattern(elev) * np.sin(elev),
                elev)
291
292         self.g0 = self.gain(0)
293         target = self.g0 + to_db(0.5)
294         self.hpbw = 2 * min(elev[np.where(self.gain(elev) < target)])
295
296     def directivity(self, elev):
297         value = 0
298         if self.p_rad is not None:
299             value = to_db(self._radiation_pattern(elev) * np.pi * 4 / self.p_rad)
300         return value
301
302     def gain(self, elev):
303         return self.directivity(elev) + self.losses
304
305     def fspl(self, r):
306         lambda_ = SPEED_OF_LIGHT / (self.freq * 1e6)
307         return to_db((lambda_ / (4 * np.pi * r)) ** 2)
308
309     def transmission_losses(self, other, distance, angle_self=0, angle_other=0):
310         gain_s = self.gain(angle_self)
311         gain_o = other.gain(angle_other)
312         fspl = self.fspl(distance)
313
314         return gain_s + gain_o + fspl
315
316     def transmit(self, receiver, distance, angle_self=0, angle_other=0):
317         return self.transmit_pwr + self.transmission_losses(receiver, distance,
                angle_self, angle_other)
318
319     def receive(self, transmitter, distance, angle_self=0, angle_other=0):
320         return self.sensibility - self.transmission_losses(transmitter, distance,
                angle_self, angle_other)
```

B.4 timed_contact.py

```
1 import numpy as np
2 from scipy.optimize import fsolve
3
4 from constants import *
5 from contact import get_angle
6 from link_distance import get_link_aperture_body
7
8
9 def get_orbit_normal_vec(lat, inc):
```

```
10      if (lat < inc) if (inc <= np.pi / 2) else (lat < np.pi - inc):
11          return np.array([
12              -np.cos(inc) * np.tan(lat),
13              np.sqrt(1 - np.cos(inc) ** 2 * (1 + np.tan(lat) ** 2)),
14              np.cos(inc),
15          ])
16      else:
17          return np.array([
18              -np.sin(inc),
19              0,
20              np.cos(inc),
21          ])
22
23
24  def align(gs, sc):
25      orbit_normal = get_orbit_normal_vec(gs.lat, sc.inc)
26      new_gs = gs.as_copy(lon=0)
27
28      raan = np.cross(np.array([0, 0, 1]), orbit_normal)
29      new_sc1 = sc.as_copy(raan=np.degrees(np.arctan2(raan[1], raan[0])))
30      r = np.linalg.inv(new_sc1.rot).dot(np.array([np.cos(gs.lat), 0, np.sin(gs.lat)]))
31      f0 = np.arctan2(r[1], r[0])
32      new_sc1.f0 = f0
33      new_sc1.update()
34
35      raan = np.cross(np.array([0, 0, 1]), orbit_normal * np.array([1, -1, 1]))
36      new_sc2 = sc.as_copy(raan=np.degrees(np.arctan2(raan[1], raan[0])))
37      r = np.linalg.inv(new_sc2.rot).dot(np.array([np.cos(gs.lat), 0, np.sin(gs.lat)]))
38      f0 = np.arctan2(r[1], r[0])
39      new_sc2.f0 = f0
40      new_sc2.update()
41
42      best_sc = max([new_sc1, new_sc2], key=lambda sc_i: np.linalg.norm(sc_i.
            get_local_position(0)))
43
44      return new_gs, best_sc
45
46
47  def rotate(r, axis, theta):
48      return r * np.cos(theta) + \
49             np.cross(axis, r, axis=0) * np.sin(theta) + \
50             axis * np.sum(axis * r, axis=0) * (1 - np.cos(theta))
51
52
53  def get_aperture(t, axis_e, axis_s, r, w_e, w_s, target):
54      return get_angle(rotate(r, axis_e, t * w_e), rotate(r, axis_s, t * w_s)) - target
55
56
57  def get_max_equatorial_comm_window(sma, min_elev):
58      return 2 * get_link_aperture_body(sma, min_elev) / np.abs(np.sqrt(BODY_MU / sma ** 3)
            - 2 * np.pi / BODY_PERIOD)
59
60
61  def get_max_comm_window(sma, lat, inc, min_elev):
62      assert (lat <= inc) if (inc <= np.pi / 2) else (lat <= np.pi - inc)
63      axis_e = np.array([[0, 0, 1]]).T
64      axis_s = get_orbit_normal_vec(lat, inc).reshape((3, 1))
65      r = np.array([[np.cos(lat), 0, np.sin(lat)]]).T
66      w_e = 2 * np.pi / BODY_PERIOD
67      w_s = np.sqrt(BODY_MU / sma ** 3)
```

```
    target = get_link_aperture_body(sma, min_elev)
    return 2 * fsolve(
        get_aperture,
        get_max_equatorial_comm_window(sma, min_elev) / 2,
        (axis_e, axis_s, r, w_e, w_s, target)
    )[0]


def get_aperture_numeric(u, gs, sc):
    gs_th, sc_th, has_sight = gs.has_sight(sc, u)
    target = get_link_aperture_body(np.linalg.norm(sc_th.position, axis=0), gs.min_elev)
    return get_angle(gs_th.position, sc_th.position) - target


def get_max_comm_window_numeric(gs, sc):
    gs_mod, sc_mod = align(gs, sc)
    start = gs_mod.has_sight(sc_mod, 0)
    if (np.linalg.norm(start[1].position) <= BODY_RADIUS) or (not np.all(start[2])):
        return 0

    t_guess = get_max_equatorial_comm_window(sc_mod.sma, gs_mod.min_elev) / 2
    u_guess = fsolve(
        lambda u, target: sc_mod.get_time(u) - target,
        np.array(2 * np.pi * t_guess / sc_mod.period),
        t_guess
    )[0]

    u1 = fsolve(get_aperture_numeric, np.array(u_guess), (gs_mod, sc_mod))[0]
    u0 = fsolve(get_aperture_numeric, np.array(-u_guess), (gs_mod, sc_mod))[0]

    return abs(sc_mod.get_time(u1) - sc_mod.get_time(u0))
```

B.5 ax25_times.py

```
from constants import AX25_T102, AX25_T2, AX25_T103, AX25_P

BIT_STUFFING = 63 / 62
OVERHEAD = 20
BYTE_TO_BIT = 8


def get_frame_time(bitrate, info_size=256, frame_count=7):
    assert frame_count <= 7
    assert info_size <= 256

    cs_time = 256 * (AX25_T102 / 100) / (2 * (1 + AX25_P))
    info_time = frame_count * BIT_STUFFING * BYTE_TO_BIT * (OVERHEAD + info_size) /
        bitrate
    rr_time = BIT_STUFFING * BYTE_TO_BIT * OVERHEAD / bitrate

    return cs_time + 2 * (AX25_T103 / 100) + info_time + (AX25_T2 / 100) + rr_time


def get_effective_bitrate(bitrate, info_size=256, frame_count=7):
    return BYTE_TO_BIT * frame_count * info_size / get_frame_time(bitrate, info_size,
        frame_count)
```

B.6 plot.py

```
import matplotlib as mpl
from matplotlib import pyplot as plt

# Graph Options
OUTPUT_PATH = r'../images/'
USE_PGF = True
PAGE_WIDTH = 6.296
SIZE_FULLPAGE = (PAGE_WIDTH, 9)
SIZE_TALL = (PAGE_WIDTH, 5)
SIZE_DEFAULT = (PAGE_WIDTH, 3)
SIZE_SHORT = (PAGE_WIDTH, 1.25)
X_RESOLUTION = 1024
N_LINES = 4

# Ranges and variables
MIN_ALTITUDE = 200  # km
MAX_ALTITUDE = 6_000  # km
GSO_ALTITUDE = 50_000  # km
FREQUENCIES = (145, 437)  # MHz
ELEV = (10, 15, 30, 45, 60)  # degrees
MIN_ELEV = min(ELEV)
LAT_AND_INC = tuple(range(0, 91, 15))
REF_ALT = MAX_ALTITUDE / 2
REF_ECC = 0.05
NUM_DAYS = 730
MAX_TRACKING = 6
MAX_TIME = 48 * 60 * 60
MAX_SCALED_AXIS = 3
MAX_DEVIATION = 20


def scale_time(interval):
    scales = {
        's': 1,
        'min': 60,
        'h': 3600,
        'd': 86400,
    }
    result = (1, 's')
    for label, scale in scales.items():
        if interval <= scale * MAX_SCALED_AXIS:
            break
        result = (scale, label)
    return result


def scale_data(interval):
    scales = {
        'b': 1,
        'B': 8,
        'KiB': 8192,
        'MiB': 8388608,
    }
    result = (1, 'b')
    for label, scale in scales.items():
        if interval <= scale * MAX_SCALED_AXIS:
            break
        result = (scale, label)
```

```
59      return result
60
61
62  class Graph(object):
63      if USE_PGF:
64          mpl.use('pgf')
65          mpl.rcParams.update({
66              'text.usetex': True,
67          })
68      plt.style.use('seaborn')
69
70      def __init__(self, filepath, graph_size=SIZE_DEFAULT, legend=False, tight=True,
            verbose=True):
71          self.path = OUTPUT_PATH + filepath + ('.pgf' if USE_PGF else '.png')
72          self.size = graph_size
73          self.legend = legend
74          self.tight = tight
75          self.fig = None
76          self.verbose = verbose
77
78      def __enter__(self):
79          if self.verbose:
80              print(f'Plotting {self.path}')
81          plt.clf()
82          self.fig = plt.gcf()
83          self.fig.set_size_inches(*self.size)
84          return self
85
86      def __exit__(self, exc_type, exc_val, exc_tb):
87          if self.tight:
88              self.fig.tight_layout()
89          if self.legend:
90              self.fig.legend()
91          self.fig.savefig(self.path)
92          plt.clf()
93          if self.verbose:
94              print(f'Finished plotting {self.path}')
95          if USE_PGF:
96              with open(self.path, 'r') as f:
97                  data = f.read()
98              with open(self.path, 'w') as f:
99                  f.write(data.replace('°', r'\degree'))
100
101
102 if __name__ == '__main__':
103     import numpy as np
104     from constants import *
105     from contact import Antenna, Contact, GroundStation, Spacecraft, get_angle
106     from link_distance import get_link_distance, get_link_aperture_sat
107     from timed_contact import align, get_max_comm_window, get_max_comm_window_numeric,
            get_max_equatorial_comm_window
108     from ax25_times import get_effective_bitrate
109
110
111     def sc_pattern(theta):
112         return (
113                 0.035395870751032045 +
114                 0.0812061856472038 * np.cos(theta) ** 6
115                 + np.cos(theta / 2) ** 8.923841918304257
116         )
```

```
117
118
119     def gs_pattern(theta):
120         theta_array = np.where(np.abs(theta) > np.pi / 2, np.pi / 2, theta)
121         return np.cos(theta_array) ** 14.811388300841896
122
123
124     # Radio configuration
125     radio_bitrate = 2400
126
127     sc_radio = {
128         'freq': FREQUENCIES[-1],
129         'vswr': 1.20,
130         'radiation_efficiency': 0.6583423465402455,
131         'radiation_pattern': sc_pattern,
132         'transmit_pwr': 30,
133         'sensibility': -120
134     }
135
136     gs_radio = {
137         'freq': FREQUENCIES[-1],
138         'vswr': 1.20,
139         'radiation_efficiency': 1,
140         'radiation_pattern': gs_pattern,
141         'transmit_pwr': 30,
142         'sensibility': -120
143     }
144
145     sc_antenna = Antenna(**sc_radio)
146     gs_antenna = Antenna(**gs_radio)
147
148     # Orbit configuration
149     iss = Spacecraft(
150         sma=6798.5e3,
151         inc=51.6430,
152         aop=116.7490,
153         raan=96.2603,
154         ecc=0.0004024,
155         f0=26.1567
156     )
157     iss.set_antenna(sc_antenna)
158
159     molnya = Spacecraft(
160         sma=((BODY_PERIOD / (4 * np.pi)) ** 2 * BODY_MU) ** (1 / 3),
161         inc=np.degrees(np.arcsin(np.sqrt(4 / 5))),
162         aop=270,
163         raan=-90,
164         ecc=0.737,
165         f0=180
166     )
167     molnya.set_antenna(sc_antenna)
168
169     sso_n_orbits = 15
170     sso_rot_period = 1.99096871e-7  # 2 * pi radians per sidereal year
171     sso_n = sso_n_orbits * 2 * np.pi / BODY_PERIOD
172     sso_sma = (sso_n ** -2 * BODY_MU) ** (1 / 3)
173     sso_inc = np.arccos(-2 * sso_rot_period / (3 * sso_n * BODY_J2 * (BODY_RADIUS /
            sso_sma) ** 2))
174
175     sso = Spacecraft(
```

```
176         sma=sso_sma,
177         inc=np.degrees(sso_inc),
178         aop=0,
179         raan=0,
180         ecc=0,
181         f0=0
182     )
183     sso.set_antenna(sc_antenna)
184 
185     polar = Spacecraft(
186         sma=sso_sma,
187         inc=90,
188         aop=0,
189         raan=0,
190         ecc=0,
191         f0=0
192     )
193     polar.set_antenna(sc_antenna)
194 
195     # Ground Stations
196     ufmg = GroundStation(-19.870682, -43.9699246, MIN_ELEV)
197     ufmg.set_antenna(gs_antenna)
198 
199     artic = GroundStation(66.5, 0, MIN_ELEV)
200     artic.set_antenna(gs_antenna)
201 
202     north = GroundStation(90, 0, MIN_ELEV)
203     north.set_antenna(gs_antenna)
204 
205     null_island = GroundStation(0, 0, MIN_ELEV)
206 
207     # Common variables
208     altitude = np.linspace(MIN_ALTITUDE, MAX_ALTITUDE, X_RESOLUTION)
209     radius = altitude * 1e3 + BODY_RADIUS  # meters
210 
211     altitude_gso = np.linspace(MIN_ALTITUDE, GSO_ALTITUDE, X_RESOLUTION)
212     radius_gso = altitude_gso * 1e3 + BODY_RADIUS  # meters
213 
214     # Gain
215     with Graph('gain_antenna', legend=True, graph_size=(0.4 * PAGE_WIDTH, SIZE_DEFAULT
            [1])) as g:
216         elev = np.linspace(-np.pi, np.pi, X_RESOLUTION)
217         ax = g.fig.subplots(subplot_kw={'projection': 'polar'})
218         ax.set_theta_direction(-1)
219         ax.set_theta_zero_location('N')
220         ax.plot(elev, sc_antenna.gain(elev), label='Transmissora')
221         ax.plot(elev, gs_antenna.gain(elev), label='Receptora')
222         ax.set_ylim([-15, 20])
223 
224     # View angle
225     with Graph('view_angle'):
226         for h in ELEV:
227             plt.plot(altitude, np.degrees(get_link_aperture_sat(radius, np.radians(h))),
                        label=r'$h = %s^\circ$' % h)
228 
229         plt.ylabel('Ângulo de visada ($^\circ$)')
230         plt.xlabel('Altitude do apogeu ($km$)')
231         plt.gca().legend()
232 
233     # Link budget
```

```
234     with Graph('free_space_path_loss', graph_size=SIZE_TALL) as g:
235         axs = g.fig.subplots(nrows=len(FREQUENCIES), sharex=True)
236 
237         for ax, f in zip(axs, FREQUENCIES):
238             ax.set_ylabel('FSPL ($db$)')
239             ax.set_title(f'{f} MHz')
240             for h in ELEV:
241                 antenna = Antenna(f)
242                 fspl = antenna.fspl(get_link_distance(radius, np.radians(h)))
243                 ax.plot(altitude, fspl, label=f'$h = %s^\circ$' % h)
244 
245         plt.xlabel('Altitude do apogeu ($km$)')
246         axs[0].legend()
247 
248     # Minimum transmit power
249     with Graph('min_transmit_power'):
250         sc_radio_lossless = sc_radio.copy()
251         sc_radio_lossless['vswr'] = 1
252         sc_antenna_lossless = Antenna(**sc_radio_lossless)
253 
254         gs_radio_lossless = gs_radio.copy()
255         gs_radio_lossless['vswr'] = 1
256         gs_antenna_lossless = Antenna(**gs_radio_lossless)
257 
258         elevation = get_link_aperture_sat(radius, np.radians(MIN_ELEV))
259         distance = get_link_distance(radius, np.radians(MIN_ELEV))
260         transmit_pwr_lossy = gs_antenna.sensibility - (
261                 sc_antenna.gain(elevation) +
262                 gs_antenna.gain(0) +
263                 sc_antenna.fspl(distance)
264         )
265 
266         transmit_pwr = gs_antenna_lossless.sensibility - (
267                 sc_antenna_lossless.gain(0) +
268                 gs_antenna_lossless.gain(0) +
269                 sc_antenna_lossless.fspl(radius - BODY_RADIUS)
270         )
271 
272         plt.fill_between(altitude, transmit_pwr, transmit_pwr_lossy, alpha=0.25)
273         plt.plot(altitude, transmit_pwr)
274         plt.ylabel('Potência mínima ($dBm$)')
275         plt.xlabel('Altitude do apogeu ($km$)')
276 
277     # Equatorial comm window
278     with Graph('comm_window'):
279         time_scale, time_label = scale_time(MAX_TIME)
280         times = np.array([
281             get_max_comm_window(sma, 0, 0, np.radians(MIN_ELEV)) for sma in radius_gso
282         ])
283 
284         gso_time = max(times)
285         gso_altitude = altitude_gso[np.where(times == gso_time)][0]
286         times[np.where(times > MAX_TIME)] = MAX_TIME * 1.1
287 
288         plt.plot(altitude_gso, times / time_scale)
289         plt.plot((gso_altitude, gso_altitude), (0, MAX_TIME / time_scale), '--', color='
                C2')
290         plt.ylabel(f'Duração da janela de comunicação (${time_label}$)')
291         plt.xlabel('Altitude do apogeu ($km$)')
292         plt.ylim(top=MAX_TIME / time_scale)
```

```
293
294     # Influence of eccentricity
295     with Graph('comm_window_ecc', graph_size=SIZE_TALL) as g:
296         axs = g.fig.subplots(nrows=2)
297
298         times_ecc = np.array([
299             [
300                 get_max_comm_window_numeric(null_island, Spacecraft(sma, REF_ECC, 0, 0,
                        aop, 0))
301                 for sma in radius
302             ]
303             for aop in (0, 180)
304         ])
305
306         time_scale, time_label = scale_time(np.max(times_ecc) - np.min(times_ecc))
307         times = get_max_equatorial_comm_window(radius, np.radians(MIN_ELEV)) / time_scale
308         times_ecc = times_ecc / time_scale
309
310         axs[0].plot(altitude, times)
311         axs[0].fill_between(altitude, times_ecc[0], times_ecc[1], alpha=0.25)
312         axs[0].set_xlabel('Altitude do apogeu ($km$)')
313         axs[0].set_ylabel(f'Duração da janela\nde comunicação (${time_label}$)')
314
315         sma = REF_ALT * 1e3 + BODY_RADIUS
316         ref_alt_time = get_max_equatorial_comm_window(sma, np.radians(MIN_ELEV)) /
                time_scale
317         ref_alt_delta = 2 * (np.max(np.abs([np.interp(REF_ALT, altitude, t) for t in
                times_ecc])) - ref_alt_time)
318         axs[0].plot(
319             (REF_ALT, REF_ALT),
320             (ref_alt_time + ref_alt_delta, ref_alt_time - ref_alt_delta),
321             '--', color='C2'
322         )
323
324         time_ref = get_max_equatorial_comm_window(sma, np.radians(MIN_ELEV))
325         ecc_axis = np.linspace(0, REF_ECC, X_RESOLUTION)
326         times_ecc = np.array([
327             [
328                 get_max_comm_window_numeric(null_island, Spacecraft(sma, ecc, 0, 0, aop,
                        0))
329                 for ecc in ecc_axis
330             ]
331             for aop in (0, 180)
332         ]) - time_ref
333         time_scale, time_label = scale_time(np.max(times_ecc) - np.min(times_ecc))
334         times_ecc = times_ecc / time_scale
335
336         axs[1].plot(ecc_axis, times_ecc[0], color='C2')
337         axs[1].plot(ecc_axis, times_ecc[1], color='C2')
338         axs[1].fill_between(ecc_axis, times_ecc[0], times_ecc[1], alpha=0.25, color='C2')
339         axs[1].set_xlabel('Excentricidade')
340         axs[1].set_ylabel(f'Variação na duração da janela\nde comunicação (${time_label}$
                )')
341
342     # Influence of inclination and latitude
343     with Graph('comm_window_lat', graph_size=SIZE_TALL) as g:
344         axs = g.fig.subplots(nrows=2)
345
346         times_inc = np.array([
347             get_max_comm_window(sma, 0, np.pi / 2, np.radians(MIN_ELEV))
```

```
348             for sma in radius
349         ])
350
351         time_scale, time_label = scale_time(np.max(times_inc) - np.min(times_inc))
352         times = get_max_equatorial_comm_window(radius, np.radians(MIN_ELEV)) / time_scale
353         times_inc = times_inc / time_scale
354
355         axs[0].plot(altitude, times)
356         axs[0].fill_between(altitude, times, times_inc, alpha=0.25)
357         axs[0].set_xlabel('Altitude do apogeu ($km$)')
358         axs[0].set_ylabel(f'Duração da janela\nde comunicação (${time_label}$)')
359
360         sma = REF_ALT * 1e3 + BODY_RADIUS
361         ref_alt_time = get_max_equatorial_comm_window(sma, np.radians(MIN_ELEV)) /
                time_scale
362         ref_alt_delta = 2 * (np.interp(REF_ALT, altitude, times_inc) - ref_alt_time)
363         axs[0].plot(
364             (REF_ALT, REF_ALT),
365             (ref_alt_time + ref_alt_delta, ref_alt_time - ref_alt_delta),
366             '--', color='C2'
367         )
368
369         time_ref = get_max_equatorial_comm_window(sma, np.radians(MIN_ELEV))
370         angle = np.linspace(0, 90, X_RESOLUTION)
371         times_max = np.array([
372             get_max_comm_window(sma, i, i, np.radians(MIN_ELEV))
373             for i in np.radians(angle)
374         ]) - time_ref
375         times = np.array([
376             get_max_comm_window(sma, 0, i, np.radians(MIN_ELEV))
377             for i in np.radians(angle)
378         ]) - time_ref
379
380         time_scale, time_label = scale_time(np.max(times_max) - np.min(times_max))
381         axs[1].plot(angle, times / time_scale, color='C2')
382         axs[1].fill_between(angle, times / time_scale, times_max / time_scale, alpha
                =0.25, color='C2')
383         axs[1].set_xlabel('Inclinação ($°$)')
384         axs[1].set_ylabel(f'Variação na duração da janela\nde comunicação (${time_label}$
                )')
385         axs[1].set_xticks(range(0, 91, 15))
386
387     # Contact histogram
388     with Graph('comm_window_hist'):
389         max_time = get_max_comm_window_numeric(ufmg, iss)
390         time_scale, time_label = scale_time(max_time)
391         times = np.linspace(0, max(max_time - 1, 0), X_RESOLUTION) / time_scale
392         max_time_scaled = max_time / time_scale
393         prob = times / (max_time_scaled ** 2 * np.sqrt(1 - (times / max_time_scaled) **
                2))
394
395         num_orbits = NUM_DAYS * BODY_PERIOD / iss.period
396         u = np.linspace(0, 2 * np.pi * num_orbits, 2 + int(num_orbits * iss.period))
397         contacts = Contact.get_contacts(u, ufmg, iss)
398
399         plt.hist([contact.duration / time_scale for contact in contacts], density=True,
                bins=16)
400         ylim = plt.ylim()
401         plt.plot(times, prob, '--', color='C2')
402         plt.ylim(ylim)
```

```
403
404         plt.ylabel(f'Densidade de probabilidade (${time_label}^{{-1}}$)')
405         plt.xlabel(f'Duração da janela de comunicação (${time_label}$)')
406
407     # Max data transfer
408     with Graph('data_hist'):
409         effective_bitrate = get_effective_bitrate(radio_bitrate)
410         max_data = max_time * effective_bitrate
411         data_scale, data_label = scale_data(max_data)
412         datarate = np.linspace(0, max(max_data - 1, 0), X_RESOLUTION) / data_scale
413         max_data_scaled = max_data / data_scale
414         prob = datarate / (max_data_scaled ** 2 * np.sqrt(1 - (datarate / max_data_scaled
                ) ** 2))
415
416         plt.hist([effective_bitrate * contact.duration / data_scale for contact in
                contacts], density=True, bins=16)
417         ylim = plt.ylim()
418         plt.plot(datarate, prob, '--', color='C2')
419         plt.ylim(ylim)
420
421         plt.ylabel(f'Densidade de probabilidade (${data_label}^{{-1}}$)')
422         plt.xlabel(f'Payload transmitida na janela de comunicação (${data_label}$)')
423
424     # Orbit
425     with Graph('orbit', graph_size=SIZE_FULLPAGE) as g:
426         labels = ('N', 'NE', 'E', 'SE', 'S', 'SW', 'W', 'NW')
427         orbits = ('ISS', 'Heliossíncrona', 'Polar', 'Molnya')
428         pairs = ((iss, ufmg), (sso, ufmg), (polar, north), (molnya, artic))
429
430         contacts = list()
431         contacts_alt = list()
432
433         for sc, gs in pairs:
434             u = np.linspace(-np.pi, np.pi, 2 + int(sc.period))
435             result = list()
436             result_alt = list()
437             gs_mod, sc_mod = align(gs, sc)
438             result.append(Contact.get_contacts(u, gs_mod, sc_mod)[0])
439
440             for lon in range(1, 91):
441                 if not gs_mod.as_copy(lon=lon).has_sight(sc_mod, 0)[2]:
442                     break
443             aperture = lon / N_LINES
444
445             for i in range(1, N_LINES):
446                 result.extend(Contact.get_contacts(
447                     u, gs_mod.as_copy(lon=i * aperture), sc_mod
448                 )[:1])
449
450                 result_alt.extend(Contact.get_contacts(
451                     u, gs_mod.as_copy(lon=-i * aperture), sc_mod
452                 )[:1])
453
454             contacts.append(result)
455             contacts_alt.append(result_alt)
456
457         n_graphs = 4
458         main_gs = plt.GridSpec(n_graphs, 1, figure=g.fig)
459         weights = (1, 2)
460
```

```
461         for i in range(n_graphs):
462             side = bool(i % 2)
463             set_gs = main_gs[i].subgridspec(1, 2, width_ratios=weights[::-1 if side else
                    1], wspace=0.15)
464 
465             ax_h = g.fig.add_subplot(set_gs[side], polar=True)
466             ax_h.set_theta_zero_location('N')
467             ax_h.set_ylim([90, 0])
468             ax_h.set_xticks(ax_h.get_xticks(), labels)
469             ax_h.set_yticks([30, 60])
470 
471             inner_gs = set_gs[not side].subgridspec(2, 1, hspace=0.1)
472 
473             ax_p = g.fig.add_subplot(inner_gs[0])
474             ax_t = g.fig.add_subplot(inner_gs[1])
475             ax_p.set(xticklabels=[])
476             ax_p.tick_params(left=False)
477             if not side:
478                 ax_p.yaxis.tick_right()
479                 ax_p.yaxis.set_label_position('right')
480                 ax_t.yaxis.tick_right()
481                 ax_t.yaxis.set_label_position('right')
482             else:
483                 ax_h.yaxis.set_label_position('right')
484 
485             time_scale, time_label = scale_time(max(
486                 [contact.sc.time[-1] - contact.sc.time[0] for contact in contacts[i]]
487             ))
488 
489             ax_p.set_ylabel(r'$P_r$ ($dBm$)')
490             ax_t.set_ylabel(r'$|{{\dot{\vec{\phi}}}_r}|$ ($° \slash s$)')
491             ax_t.set_xlabel(f'Tempo $({time_label})$')
492 
493             ax_h.set_ylabel(orbits[i], labelpad=40, fontweight='bold')
494 
495             max_tracking = list()
496             tracking_limit = False
497             for j, contact in enumerate(contacts[i]):
498                 ax_h.plot(contact.azimuth, np.degrees(contact.elevation), color=f'C{j}')
499                 if j > 0:
500                     contact_alt = contacts_alt[i][j - 1]
501                     ax_h.plot(contact_alt.azimuth, np.degrees(contact_alt.elevation),
                            color=f'C{j}')
502                 t0 = min(contact.sc.time[np.where(contact.elevation == max(contact.
                        elevation))])
503                 sc_time = (contact.sc.time - t0) / time_scale
504                 ax_p.plot(sc_time, contact.power)
505                 tracking = np.degrees(np.linalg.norm(contact.tracking, axis=0))
506                 ax_t.plot(sc_time, tracking)
507                 max_tracking.append(max(tracking[np.where(tracking < MAX_TRACKING)]))
508                 if max(tracking) > MAX_TRACKING:
509                     tracking_limit = True
510             if tracking_limit:
511                 ax_t.set_ylim([0, 1.25 * max(max_tracking)])
512 
513     with Graph('gain_sensibility'):
514         max_view_angle = get_link_aperture_sat(radius[0], np.radians(MIN_ELEV))
515         theta = np.linspace(0, max_view_angle, X_RESOLUTION // 4).reshape((X_RESOLUTION
                // 4, 1))
516         delta_theta = np.linspace(0, np.radians(MAX_DEVIATION), X_RESOLUTION)
```

```
517         gains = sc_antenna.gain(theta + delta_theta.reshape((1, X_RESOLUTION))) -
                sc_antenna.gain(theta)
518         plt.fill_between(np.degrees(delta_theta), np.max(gains, axis=0), np.min(gains,
                axis=0), alpha=0.25)
519         plt.plot(np.degrees(delta_theta), sc_antenna.gain(delta_theta) - sc_antenna.gain
                (0), label='Transmissora')
520         plt.plot(np.degrees(delta_theta), gs_antenna.gain(delta_theta) - gs_antenna.gain
                (0), label='Receptora')
521         plt.ylabel('Variação no ganho ($dBm$)')
522         plt.xlabel('Erro no apontamento ($^\circ$)')
523         plt.gca().legend()
524
525     with Graph('tracking_sensibility', graph_size=SIZE_TALL) as g:
526         contact = contacts[0][0]
527         axs = g.fig.subplots(nrows=3, sharex=True)
528
529         time_scale, time_label = scale_time(contact.sc.time[-1] - contact.sc.time[0])
530         sc_time = contact.sc.time / time_scale
531
532         tracked_elev = contact.elevation
533         az_avg = (contact.azimuth[-1] + contact.azimuth[0]) / 2
534         slope = np.sign(contact.azimuth[-1]) * contact.sc.time * np.radians(MAX_TRACKING)
535         tracked_az = np.where(np.abs(contact.azimuth - az_avg) < np.abs(slope), contact.
                azimuth, slope + az_avg)
536
537         axs[0].plot(sc_time, np.degrees(contact.elevation), label='Satélite')
538         axs[0].plot(sc_time, np.degrees(tracked_elev), '--', color='C2', label='Rastreio'
                )
539         axs[0].set_ylabel(r'Elevação ($^\circ$)')
540         axs[0].legend()
541
542         axs[1].plot(sc_time, np.degrees(contact.azimuth))
543         axs[1].plot(sc_time, np.degrees(tracked_az), '--', color='C2')
544         axs[1].set_ylabel(r'Azimute ($^\circ$)')
545
546         sc_pos = np.array([
547             np.cos(contact.azimuth) * np.cos(contact.elevation),
548             np.sin(contact.azimuth) * np.cos(contact.elevation),
549             np.sin(contact.elevation)
550         ])
551
552         tracked_pos = np.array([
553             np.cos(tracked_az) * np.cos(tracked_elev),
554             np.sin(tracked_az) * np.cos(tracked_elev),
555             np.sin(tracked_elev)
556         ])
557
558         angles = get_angle(sc_pos, tracked_pos)
559         angles[np.where(np.isnan(angles))] = 0
560         axs[2].plot(sc_time, np.degrees(angles))
561         axs[2].set_ylabel(r'Erro ($^\circ$)')
562         axs[2].set_xlabel(f'Tempo $({time_label})$')
```