ADRIANO CÉSAR DE SOUSA PEREIRA
JOÃO PEDRO BINELI ALVES
KAYQUE BATISTA DE OLIVEIRA
KEVIN COHIM HEREDA DE FREITAS MARINHO

ANÁLISE DA INTEGRIDADE DOS SUBSISTEMAS E COMPORTAMENTO ORBITAL DO CUBESAT EDUCACIONAL, ICARUS-1

São João da Boa Vista - SP
2023

RESUMO

O presente trabalho se trata do relatório da 1ª Olimpíada Brasileira de Satélites (OBSAT), etapa regional (3ª fase) e nacional (4ª fase). O objetivo é o de estabelecer uma nova missão para um lançamento suborbital do Cubesat educacional, ICARUS-1da 4ª fase, sendo ainda o foco o estudo da radiação térmica e demais comportamentos atmosféricos e orbitais, tais como: aceleração, velocidade, temperatura, pressão absoluta, pressão relativa, altitude, latitude, longitude, comunicação RF na faixa de rádio amador, giroscópio, acelerômetro, vida útil de bateria. Cujo resultado será utilizado para servir de referencial teórico e prático para outros projetos de nanosatélites. Para a missão proposta, o estudo da radiação térmica, se faz necessário um isolamento térmico a fim de refletir parte da radiação térmica incidente pelo sol, albedo terrestre e infravermelho. A radiação térmica só depende da superfície do material, ou seja, a relação entre emissividade e absortividade. Para o estudo dos dados coletados, será utilizado o método da análise por elementos finitos, o mesmo empregado para as simulações estruturais presentes no relatório. A estrutura do ICARUS-1 foi feita utilizando Alumínio 7075 (aeronáutico), o qual possui alta resistência e um pouco mais leve que os alumínio convencionais. Foram feitas análises modais e em resposta a frequência de 0-233Hz, entre as quais demonstraram que a estrutura atende às exigências do edital, com deslocamento máximo de 0,147mm. Os testes de gravidade, onde foi aplicado uma condição de 30G, resultaram em um deslocamento máximo de 0,053mm, ou seja, adequado às forças exercidas pelo veículo lançador.

Palavras-chave: Cubesat; OBSAT; Radiação Térmica.

SUMÁRIO

1. INTRODUÇÃO

O presente trabalho tem como objetivo o relatório da missão da 3ª fase da Olimpíada Brasileira de Satélites (OBSAT) e melhorias implementadas para a 4ª fase. A missão proposta se dá ao estudo do comportamento térmico a fim de auxiliar no desenvolvimento de um Subsistema de Controle Térmico. Para atender ao objetivo principal, é necessário realizar simulações computacionais e testes experimentais. Portanto, a solução do objetivo principal se faz necessário, também, a compreensão das fontes de calor e como afetam um corpo em orbita terrestre por radiação solar e condução. Os objetivos secundários incluem o monitoramento da temperatura, pressão, nível de bateria, posicionamento angular, aceleração e informações de carga útil. Ambos se devem aos critérios avaliativos da OBSAT.

Além dos requisitos de missão e critérios avaliativos, a equipe optou por desenvolver um Cubesat com componentes acessíveis no varejo a fim de utilizá-lo como ferramenta educacional para alunos, professores e pesquisadores. Partindo do exposto, o Cubesat ICARUS-1estabelece uma base para futuros projetos, incluindo a participação nas futuras edições da OBSAT e fornecendo margem para melhorias.

2. TRANSFERÊNCIA DE CALOR

A transferência de calor é feita por diferentes meios, como: condução, convecção, radiação. Os tópicos em diante descrevem de forma simplificada os fenômenos e sua influência em um satélite na órbita baixa terrestre.

2.1 Condução

A condução térmica refere-se a capacidade de um corpo conduzir (ou

transferir) calor através dos materiais. Dado o exposto, um satélite está sujeito a troca de calor entre a estrutura, desde os elementos ativos como passivos, estes produzirão troca de calor por condução.

Para o ICARUS-1, não há elementos ativos para controle de temperatura, apenas passivos, utilizando materiais que suportam altas temperaturas para manter os componentes dentro da faixa de operação. São utilizados dessa forma, isolamentos multicamada (MLI: *Multilayer Insulation*), e materiais com alto coeficiente de condutividade térmica como o alumínio comumente usado na estrutura, além de outros materiais.

2.2 Convecção

A convecção se dá a capacidade de transferir calor através de um meio líquido ou gasoso, predominantemente presente na atmosfera terrestre. Nesse caso, elementos que produzem calor, além de transferir por condução, irá emitir ondas de calor através dos gases/líquidos.

A 3ª fase da OBSAT predominou o fenômeno da convecção internamente e externamente à estrutura do satélite, tendo em vista a presença da atmosfera a qual a temperatura externa varia (não linear) com a altitude.

2.3 Radiação térmica

A radiação térmica presente no satélite, provém da radiação solar, esta se propaga na forma de onda eletromagnética e não depende de um meio para sua condução. Ou seja, não depende da convecção.

Além do exposto, há condução de calor pois a radiação térmica abrange diversos comprimentos de onda, sendo o comprimento da luz visível aquele que transfere calor. Porém em atmosfera terrestre, estão presentes: radiação solar direta, albedo terrestre, infravermelho. Sendo o infravermelho irradiado pela terra e o albedo a radiação solar refletida na terra.

Por se tratar de um tema amplo, destaca-se que uma abordagem para amenizar o impacto da radiação térmica é o acabamento da superfície do material, pois só depende da superfície.

A superfície do material deve ser reflexiva o suficiente para poder refletir a radiação térmica para outra direção, contudo sempre haverá uma parcela da radiação que atravessa a estrutura de um corpo, pois depende da emissividade da

superfície. A emissividade varia de 0 a 1 e materiais com superfícies anodizadas, polidas, tendem a ser uma ótima opção.

3. METODOLOGIA

A execução do projeto partiu da organização das ideias através da plataforma gratuita de elaboração de projetos, Miro. O mapa conceitual apresentado no apêndice A refere-se ao projeto do Cubesat ICARUS-1.

Não é possível analisar com detalhe cada elemento da figura, porém constam: datasheets; vídeos explicativos que serviram de referencial teórico para a programação; links dos produtos ou componentes; sites diversos; “pinout” (ou pinagem) dos periféricos utilizados. Toda a informação contida no mapa dividida por projeto estrutural a esquerda e projeto eletrônico à direita. As pesquisas foram realizadas por todos os membros da equipe, incluindo sugestões do orientador Daniel E. Razera.

A partir do exposto, foram feitas as programações individuais de cada componente, complementando-os aos poucos a medida em que funcionavam. Cada membro da equipe ficou responsável por estudar e programar algum componente. Ainda, houve reuniões presenciais e a distância para execução do projeto.

O orçamento dos componentes foi feito por Adriano C. S. Pereira e houve a contribuição de todos os integrantes da equipe, incluindo o orientador.

Para a estrutura foram feitas modelagens 3D através do software Autodesk Inventor Professional, versão estudantil e desenvolvidas por Adriano C. S. Pereira. Impressão feita em PLA (3ª fase), disponível no IFSP - SBV e Alumínio 7075 (4ª fase), comprado pelos integrantes da equipe.

A montagem da estrutura em PLA e alumínio foram realizadas por Kayque B. Oliveira e Adriano C. S. Pereira. A impressão 3D foi feita através da impressora do orientador Daniel E. Razera, e estrutura em alumínio, parte com ferramentas próprias de Kayque B. Oliveira e Adriano C. S. Pereira durante o período de recesso acadêmico. Após o recesso, parte da estrutura atualizada foi feita no laboratório de mecânica do IFSP – SBV.

Para a programação final do satélite, compilada por Adriano C. S. Pereira, foram feitas duas abordagens diferentes para a 3ª e 4ª fase. A 3ª fase recebeu uma programação com o intuito de otimização do consumo de energia, mas realizando menos leituras devido a função hibernar (Deep-Sleep) empregada. A programação foi feita através do software

gratuito IDE Arduino.

A 4ª fase se trata de um lançamento suborbital, portanto há a necessidade de leituras com menor intervalo, nesse caso houve a troca da bateria por uma com mais carga e o computador de bordo opera continuamente com otimização do processo, utilizando 2 núcleos do microcontrolador.

Com o processo de prototipagem concluído, foram feitas as uniões dos componentes eletrônicos em placas de circuito ilhadas disponibilizadas pelo IFSP - SBV. O aluno Kayque B. Oliveira e Adriano C. S. Pereira participaram do processo.

Com o objetivo principal previamente estabelecido, foi feito o estudo da radiação térmica por simulação, em uma estrutura cuboide utilizando a método da Análise por Elementos Finitos através do software Ansys, versão estudantil. Vale destacar que não foi possível refinar a análise devido às limitações de elementos da versão estudantil, pois o estudo da Radiação Térmica por Elementos Finitos é limitado a poucos softwares. Portanto, partiu-se do aprofundamento do estudo por artigos, livros e sites que debatem o tema a fim de estabelecer uma hipótese para a solução do problema proposto. O estudo foi realizado pelo Adriano C. S. Pereira e teve orientação do Prof. Dr. Emerson dos Reis do IFSP, o qual ofertou a disciplina optativa do Método de Elementos Finitos.

A partir de pesquisas, descobriu-se que o melhor candidato para medir a influência da radiação térmica: o termistor. Contudo, se trata de um componente analógico e é preciso ser calibrado utilizando um padrão cuja precisão seja superior. Os termistores foram calibrados no IFSP, utilizando um termômetro de bulbo como padrão e divisor de tensão nos termistores para ler a variação de tensão conforme variava a temperatura. O processo de calibração foi feito por Kayque B. Oliveira e Adriano C. S. Pereira.

Foi feito o isolamento térmico como controle passivo de temperatura, para uma primeira abordagem na solução do problema da radiação térmica. A hipótese foi desenvolvida por Adriano C. S. Pereira e manufatura com auxílio de Kayque B. Oliveira.

As melhorias do projeto, bem como o estudo da radiação térmica, foi realizada em partes, tendo em vista alguns desvios nas dimensões que tornam a estrutura inadequada para o lançamento, este problema ser solucionado até a data de lançamento no Centro de Lançamento de Alcântara. A simulação da radiação térmica foi executada novamente, porém através do software Autodesk Inventor com o complemento de Análise por Elementos Finitos, Autodesk Nastran.

4. MATERIAIS

A seguir consta a listagem na tabela 4.1 dos materiais utilizados na 3ª fase com orçamento, quantidade e descrição.

Tabela 4.1: Lista de materiais, ICARUS-1.

Qtd.	Componente	Descrição	Valor
1	ESP32	Microcontrolador ESP32 NODEMCU de 38 pinos	R$ 44,61
1	MPU6050	Giroscópdio e Acelerômetro com 6 eixos	R$ 32,36
1	BMP280	Módulo sensor de temperatura, pressão, altitude	R$ 27,73
1	NEO6MV2	GPS UBLOX NEO6M-V2	R$ 34,67
2	Bateria	3,7V, 1020 mAh, íon-lítio, bateria de celular	R$ 28,27
3	Placas	Circuito ilhado 10x10 cm	R$ 51,35
1	Leitor de cartão	Módulo leitor de cartão micro SD	R$ 27,94
1	Porcas	Porcas autotravantes M3	R$ 33,50
4	Espaçadores	Espaçador hexagonal M3x18 mm, pacote com 10	R$ 116,30
1	Multiplexador	16x1 para leitura do termistor	R$ 28,70
2	TP4056	Circuito de recarga até 1	R$ 27,74
1	Micro SD	Cartão micro SD 2GB	R$ 36,40
1	Parafusos	Allen cabeça chata M3x20, pacote com 50	R$ 44,90
	TOTAL		**R$ 534,47**

Fonte: elaborado pelo autor.

5. SUBSISTEMAS

A seguir são descritos os subsistemas do ICARUS-1 para a 4ª fase, destacando as melhorias que foram feitas em relação à 3ª fase.

5.1 Computador de bordo:

Assim como na 3ª fase, está sendo utilizado a mesma placa de desenvolvimento: ESP32 NODEMCU 38 pinos. O microcontrolador faz a comunicação Wi-Fi quando há necessidade, mas não será utilizado na 4ª fase. Foi mantido este microcontrolador devido a sua gama de aplicações, antes utilizado com hibernar, mas sem a possibilidade do uso dos 2 núcleos. Na

versão atual do código serão utilizados ambos os núcleos.

5.2 Telemetria:

Para a aquisição dos dados, estes serão armazenados em cartão de memória e cada linha será transmitida por radiofrequência através do protocolo LoRa (chip SX1278) no módulo ED32-433T20DC, na frequência de 433 MHz. Foi escolhido este módulo devido ao seu consumo médio de potência ser próximo de 1W, metade do comumente encontrado em placas de comunicação RF para Cubesats;

5.3 Suprimento de Energia:

Houve a substituição da bateria e circuito de recarga utilizado na 3ª fase. Anteriormente foi feita a associação em série incorreta (não utilizava BMS), e com 2 circuitos de recarga. A melhoria atual conta com uma única bateria de 5000 mAh, 3,7V e com um circuito de recarga de tensão de saída ajustável J5019 para até 24V, sendo este ajustado para 7V, pois 5V é a tensão mínima necessária para o regulador de tensão AMS1117 reduzir a tensão para 3,3V necessários no ESP32, este regulador encontra-se soldado na placa de desenvolvimento e permite uma entrada de 5-15V. Contudo, elevar a tensão de saída do circuito faz com que a carga da bateria seja reduzida. A substituição da bateria reduziu 11g do sistema de potência.

5.4 Carga Útil:

5.4.1. Sensoriamento:

Os sensores utilizados são os mesmos da 3ª fase: BMP280, MPU6050, NTC 10k, porém a posição do termistor situado na bateria, mudou para a face inferior da estrutura;

5.4.2. Rastreio:

Para o rastreio será utilizado o módulo GPS em conjunto com a comunicação RF que fará a transmissão das informações de latitude e longitude do satélite. Foram vistos problemas com o módulo na 3ª fase, portanto foi necessária a substituição do módulo e antena.

6. ESTRUTURA

6.1 Modelagem 3D:

O modelo 3D no apêndice E representa a estrutura como encontra-se

fisicamente. Parte dos componentes eletrônicos utilizados são ilustrativos e não fazem parte das simulações realizadas.

6.2 Manufatura:

O apêndice F apresenta a manufatura das peças em alumínio. Com o auxílio da esmerilhadeira e serra vertical, foram feitos os cortes dos trilhos, face superior e inferior, esqueleto interno. Todas as chapas de alumínio 7075 cortadas possuem 100x100mm com espessura de 1mm, aproximadamente, com erro de $\pm 0{,}1$mm. A massa de cada chapa, antes do corte, possui 28g e erro de ± 1g.

6.3 Blindagem Eletromagnética:

Na 3ª fase foi utilizada uma gaiola de Faraday, aterrada no microcontrolador, cobrindo a região dos sensores. A região descoberta situa-se na placa superior, onde encontra-se a placa de desenvolvimento, antena do GPS e leitor de cartão. A gaiola de Faraday foi removida para a 4ª fase. A própria estrutura, apesar de não estar aterrada devido a posição da antena ser interna, exclui a possibilidade de uma gaiola de Faraday. Uma possível configuração que permite a reinclusão da gaiola de Faraday é no caso de a antena ser externa e depende do espaço disponível no "dispenser" do veículo lançador.

6.4 Isolamento Térmico:

Observa-se no apêndice G o isolamento térmico, o qual foi feito de modo a refletir a radiação térmica incidente na estrutura, mas também suportar altas temperaturas. Foram feitas uma camada de fita Kapton (ou filme de poliamida), 3 camadas de Mylar (ou filme de poliéster), esqueleto interno da estrutura, 3 camadas de Mylar e fita Kapton, respectivamente. O mesmo processo aplicado nas faces superior e inferior.

Para as faces superiores e inferiores, foram adicionados Polietileno expandido, sendo que a face em contato com o alumínio possui uma camada de Kapton. O intuito é manter a temperatura na chapa, prevenindo a transferência de calor por convecção e condução. Sendo o efeito da convecção minoritário em elevadas altitudes. A condução se faz das placas aos espaçadores e fixação da antena do GPS até as placas de circuito. A reflexão da radiação térmica está presente internamente também, nas faces das placas de circuito superior e inferior, reduzindo

a transferência por onda eletromagnética até a face central, onde encontram-se os sensores.

6.5 Estrutura final:

Após serem desenvolvidas as melhorias, descritas nos tópicos anteriores, o apêndice H exibe a estrutura completa: estrutura interna com placas de circuito e estrutura externa com isolamento térmico.

A região dos furos torna possível o ajuste das dimensões, com desvio médio de 0,5mm e erro de $\pm 0{,}1$mm. As dimensões verticais excedem o exigido no edital em 3,3mm, sendo necessária a correção até a data de lançamento do satélite.

Além do exposto, foi feito chanfro na região dos furos (não visível no modelo 3D), ou seja, a dimensão até os furos pode exceder entre 0,3-1,0mm, devido a cabeça do parafuso ficarem parcialmente externos à estrutura.

7. ELETRÔNICA

7.1 Placas de Circuito:

Os componentes foram soldados nas placas ilhadas, havendo alterações apenas na placa da bateria. Para recapitular o que foi feito na 3ª fase, a figura no apêndice I refere-se a placa superior com computador de bordo, leitor de cartão, GPS e botão liga/desliga. O verso da placa faz as ligações com os pinos do ESP32 e demais componentes nas placas inferiores, além de possuir o módulo do GPS soldado no verso.

Em sequência a placa central encontra-se os sensores BMP280, MPU6050, MUX 16x4 e terminais de conexão em que são conectados os 6 termistores.

O circuito de comunicação, visto na última figura, situa-se no verso do circuito da bateria, este tratado no subtópico 7.2. Estão presentes o módulo LoRa, antena, circuito de recarga J5019 e termistor avulso, este fixado na figura à direita, referente a face inferior do Cubesat.

7.2 Isolamento:

Observa-se no apêndice I, nas placas de circuito superior e inferior, a presença da fita térmica reflexiva, com o objetivo de proteger o circuito e refletir uma parcela do que a radiação térmica penetra a estrutura.

Foi feito o isolamento da bateria de modo a proteger os circuitos superiores, onde há a presença do Polietileno expandido entre a placa central (sensoriamento) e subsistema de potência no apêndice J.

O polietileno expandido ajuda a transferir calor e caso ocorra vazamento ou inflamação, o material sofre os danos antes de poder prejudicar o circuito. Além do exposto, a bateria recebeu camadas de fita Kapton próxima ao circuito de conexão com saídas dos polos positivo e negativo.

8. PROGRAMAÇÃO

A programação do satélite parte do uso de bibliotecas de sensores e programação própria para os termistores, o qual realiza leitura analógica. O fluxograma do código no apêndice K representa o processo de leitura, armazenamento e transmissão dos dados.

O processo de envio dos dados, visto no apêndice A, é feito através do segundo núcleo do microcontrolador, descrito na função “loop2”, a fim de evitar atrasos por envio dos dados, este podendo durar até 1,2 segundos para a transmissão.

Já o núcleo principal, função “loop” por padrão, executa o processo de leitura dos sensores, GPS, nível de bateria e escrita dos dados no cartão de memória.

9. TELEMETRIA

A aquisição dos dados é feita por um rádio amador de banda dupla com faixa de frequência de 433MHz, compatível com a frequência de transmissão do módulo LoRa do satélite. A antena utilizada na estação de solo é do tipo yagi com ganho de 8db e impedância de 50Ω. A antena acoplada no satélite é do tipo omnidirecional com impedância de 50Ω. O alcance máximo da antena do satélite varia até 3km, distância suficiente para o enlace com a antena de solo.

Os dados recebidos do satélite são armazenados em computador através do GNURadio, plataforma gratuita de comunicação por rádio frequência.

10. TESTES 3ª FASE

Foram feitos testes presenciais de robustez mecânica ao aplicar vibrações no satélite. Todos os componentes permaneceram operantes. A massa total do Cubesat na 3ª fase era de 439g.

As medidas das dimensões estavam próximas da tolerância com desvios próximos

de 1mm. O sistema permaneceu operante após os testes térmicos, em que foi utilizado gelo seco para atingir temperaturas abaixo de 0 °C, inicialmente com -62°C.

Ainda, o texte de robustez eletromagnética identificou transmissão de 433 MHz, porém não havia placa de comunicação na 3ª faze, provavelmente foi o chip RF do ESP32 sendo acionado quando ligada a placa.

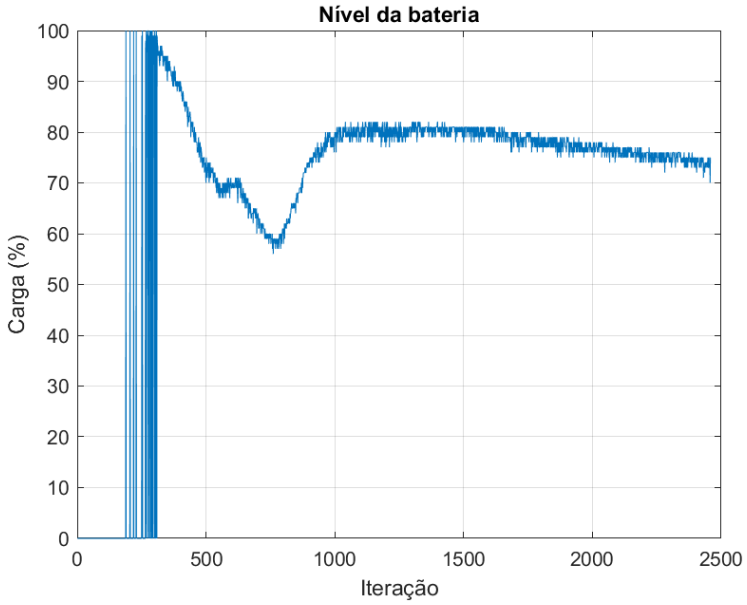
Não foi possível estabelecer conexão com o receptor local (nenhuma equipe conseguiu), mas foi possível com o servidor de testes da OBSAT. O leitor de cartão não armazenou os dados, mas o problema foi corrigido até o dia do lançamento que havia sido adiado (2º dia da LASC).

11. RELATÓRIO DE VOO 3ª FASE

Os dados coletados e armazenados em cartão de memória foram apurados graficamente como consta nas figuras a seguir. Inicialmente com dados obrigatórios da OBSAT. O timestamp está definido para 10 segundos por leitura, ou seja, o eixo X de cada gráfico refere-se a iteração realizada na leitura, totalizando aproximadamente 7 horas de operação.

Os dados originais estão disponíveis no Github e o acesso consta no apêndice.

Figura 11.1 – Nível de bateria



Fonte: elaborado pelo autor.

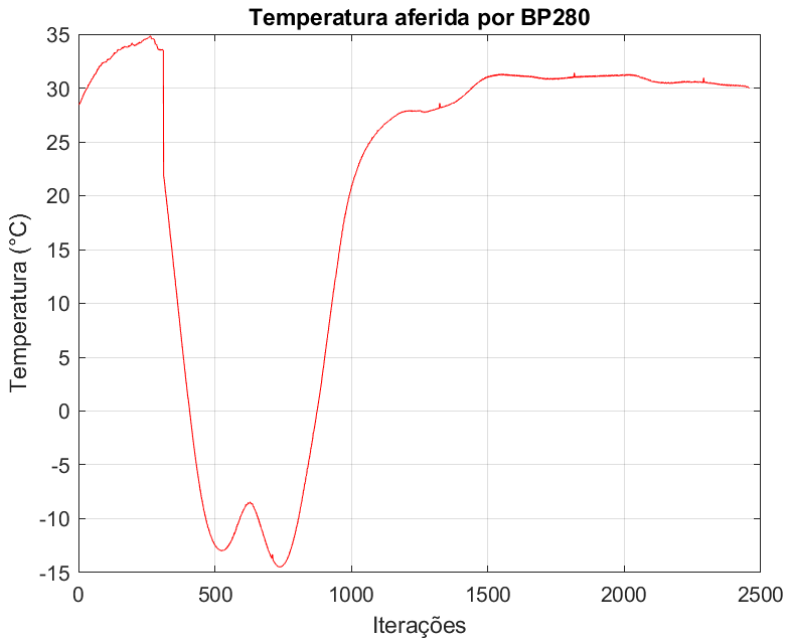
Observa-se a partir da iteração 1000 o decaimento da carga da bateria e o que

justifica os valores inconsistentes antes desse período se deve à etapa de espera até o lançamento, o qual houve alguma anomalia no funcionamento do satélite, este não funcionando por um bom período e tornou a funcionar quando estava em 9 km de altitude.

Apesar do problema observado, os valores são coerentes a partir da iteração 1000 e comprova a eficiência do modo hibernar, sendo este fundamental para missões de longa duração, inclusive quando em órbita. Portanto, é fundamental a otimização do consumo de energia para garantir a longevidade da bateria. Além do exposto, o decaimento da bateria também se deve às baixas temperaturas influenciarem na corrente elétrica.

Em seguida é visto na figura 11.2 a temperatura medida pelo BP280.

Figura 11.2 – Temperatura medida pelo BMP280

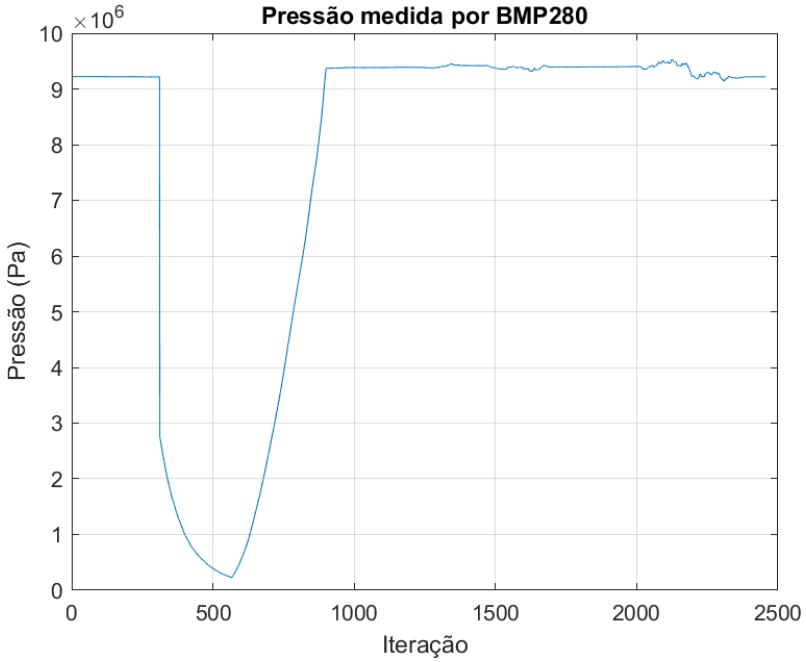


Fonte: elaborado pelo autor.

Apesar do que ocorreu com a leitura da bateria, o módulo BMP280 foi um dos componentes que permaneceu operante durante todo o período e são observadas temperaturas próximas de -15°C devido à subida da sonda para altitudes elevadas, onde a temperatura pode ser baixa.

A figura 11.3 reforça o exposto, pois refere-se à variação de pressão.

Figura 10.3 – Pressão medida por BMP280

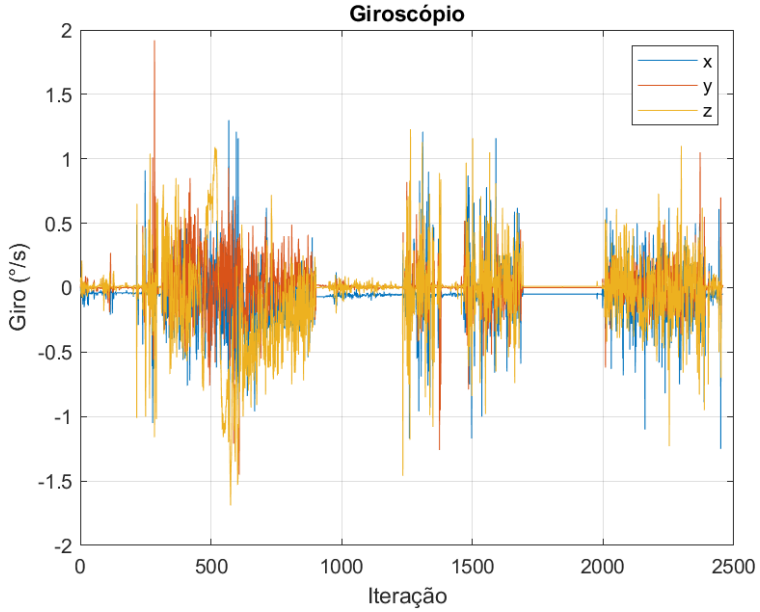


Fonte: elaboração própria.

Quanto maior a altitude, menos será a pressão devido à concentração da matéria estar presente próxima da terra. A temperatura também está relacionada a pressão, o que reforça o deslocamento vertical do Cubesat. Ainda, é possível comparar esses dados com valores padrões atmosféricos aferidos por diversos órgãos. Mas vale destacar que o sensor utilizado não deveria realizar leituras de pressão acima dos 9000 metros, de acordo com o datasheet.

A seguir são descritos os valores de rotação e aceleração medidos pelo módulo MPU6050.

Figura 11.4 – Dados dos 3 eixos do giroscópio.

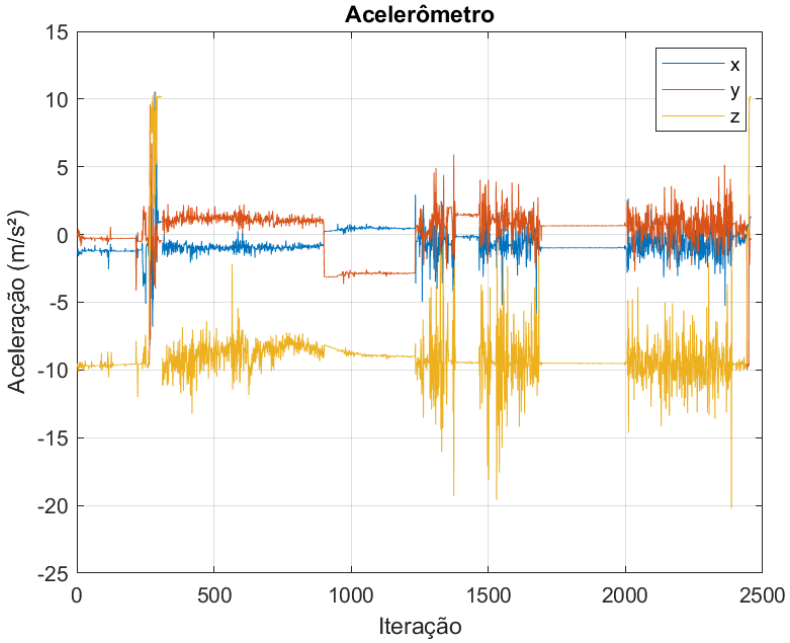


Fonte: elaboração própria.

Durante o lançamento da sonda, houve um acidente em que a sonda sofreu um choque com uma estrutura em solo e o ICARUS-1 caiu no chão. O período que representa a queda pode ser facilmente visto pelo pico de rotação no eixo Y próximo de 2°/s (limite do módulo). As demais oscilações são provenientes do deslocamento e operação dos orientadores no local do evento e a partir da iteração 300, aproximadamente, se tem o deslocamento natural após o choque. Os valores altos de oscilação podem ser justificados por ventos que são fortes na superfície terrestre, mudança de direção, e próximo de 1500 iterações, a queda da sonda. O módulo não possui filtros, naturalmente desenvolve oscilações indesejáveis.

É importante o uso do giroscópio para controle de apontamento, ou referenciamento da orientação do satélite em órbita. Em sequência, como complemento do giroscópio, destaca-se o acelerômetro.

Figura 11.5 – Dados dos 3 eixos do acelerômetro.

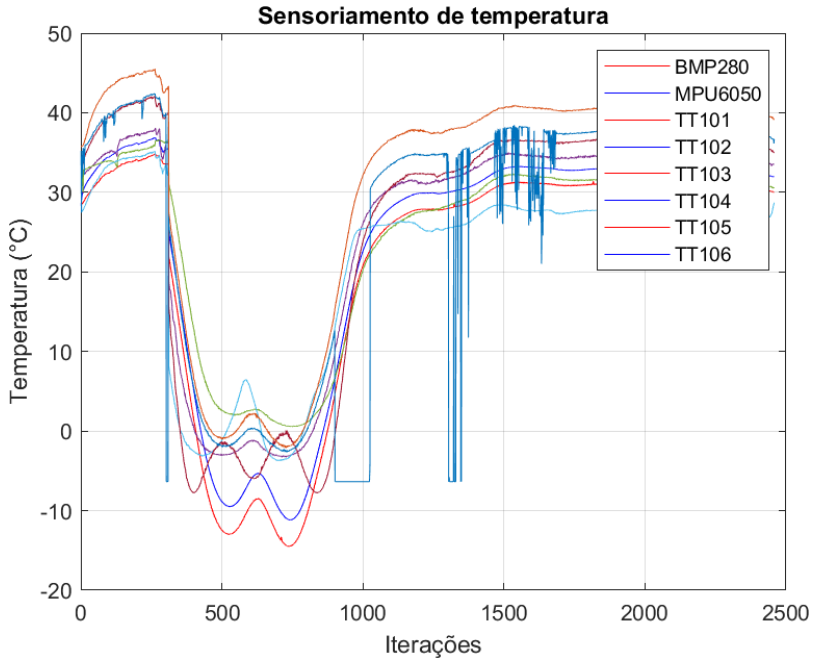


Fonte: elaboração própria.

Observa-se na leitura próxima de 250 iterações o que foi visto no gráfico da figura 10.4, o momento em que ocorre o choque. As mesmas oscilações e fatores que influenciam o giroscópio, tendo em vista a similaridade do comportamento gráfico, contudo o acelerômetro indica a aceleração no eixo, sendo maiores em torno da iteração de 1500.

Dado o exposto, serão vistos em seguida os dados coletados na Carga Útil do satélite.

Figura 11.6 – Sensoriamento de temperatura.



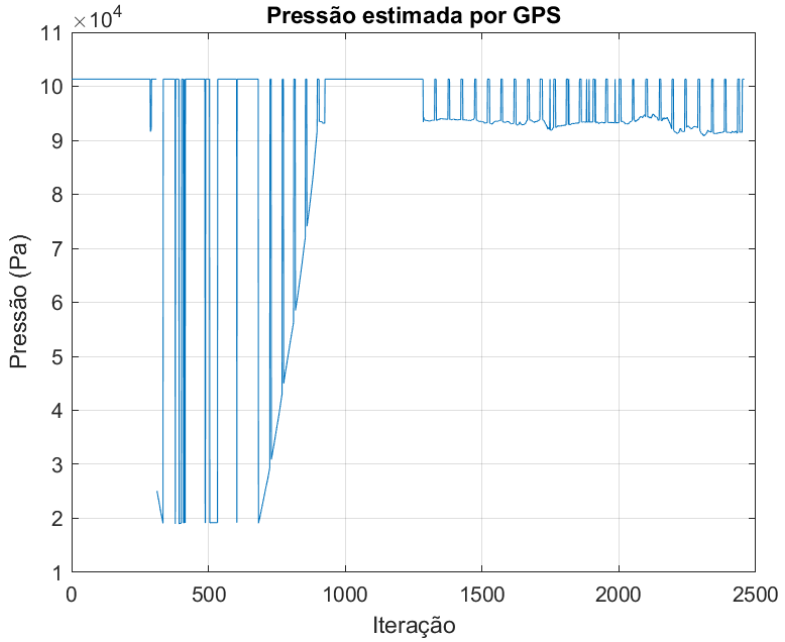
Fonte: elaborado pelo autor.

É fácil observar que as leituras tendem para valores próximos, porém com desvios elevados entre si, destacando os 2 sensores digitais, MPU6050 e BMP280. Se faz evidente que os termistores operaram de forma anômala, como visto pelo TT106 cuja leitura falha por volta de 1300-1700 iterações.

É importante notar que cada sensor está situado em posições distintas, o que justifica a variação de temperatura entre si. Apesar da leitura ser aproximada, os valores são confiáveis e próximos do esperado.

Em sequência, dado a preocupação com o possível mal funcionamento do BMP280, observa-se na figura 11.7 a seguir a estimação da pressão feita pelo GPS.

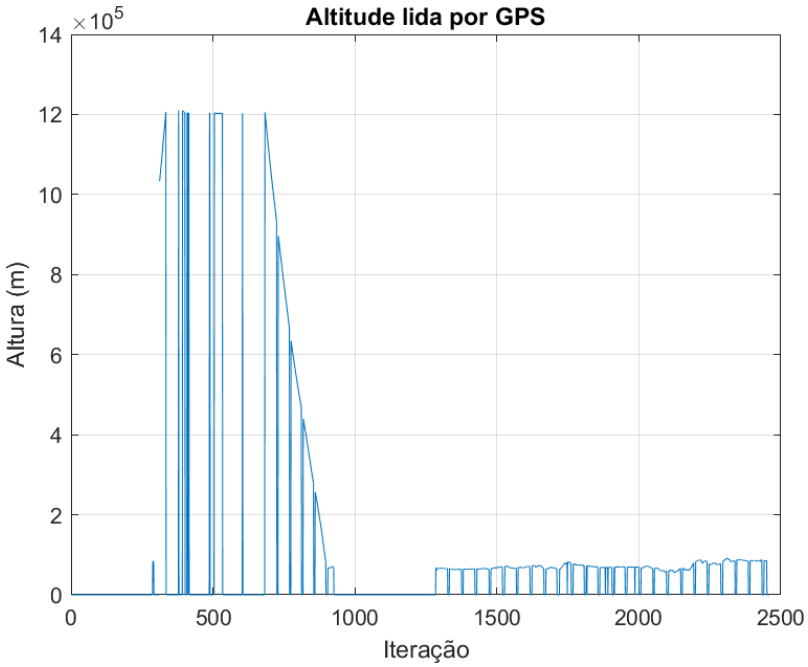
Figura 11.7 – Pressão estimada por GPS



Fonte: elaborado pelo autor.

A pressão tende às leituras feitas pelo BMP280, contudo permanecem constantes em 20000 Pa, o que não faz sentido. Para complementar, observa-se a figura 11.8 a seguir.

Figura 11.8 – Altitude lida por GPS

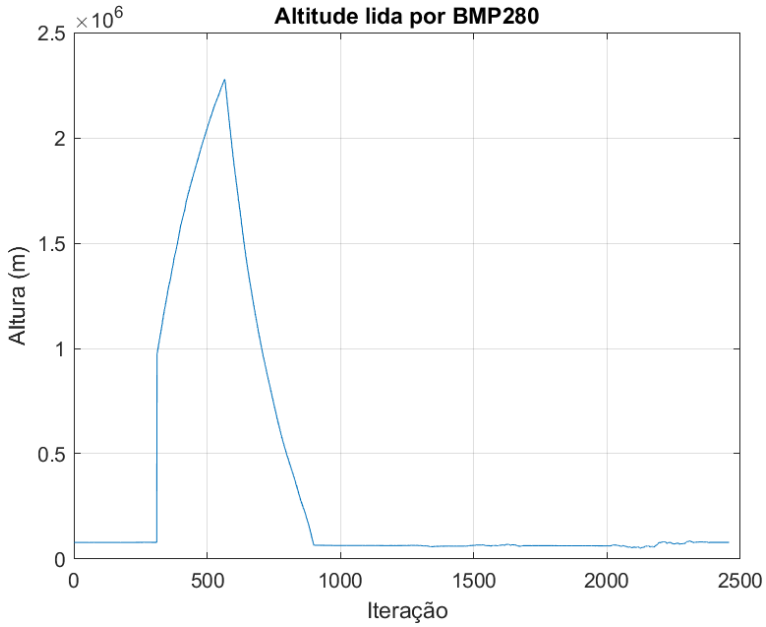


Fonte: elaborado pelo autor.

A leitura da altitude aproxima-se dos 12 km e permanece constante, com exceção dos momentos de inatividade de sinal. Significa que o GPS está lendo de forma incoerente

a altura. Ainda, para reforçar o exposto, observa-se a altitude lida pelo BMP280 na figura 11.9.

Figura 11.9 – Altitude lida por BMP280

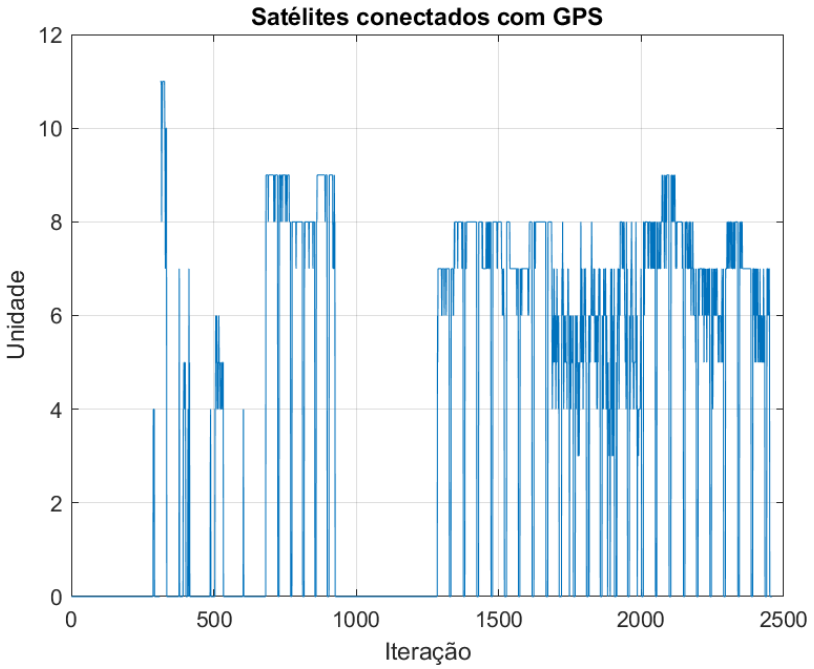


Fonte: elaborado pelo autor.

Ao contrário do GPS, a altura medida é consistente com o previsto no edital e organização no local do evento, ou seja, o balão atingiu altitude máxima de aproximadamente 22 km. Portanto, o GPS não funcionou corretamente, mas sim o BMP280.

Em seguida são vistas as conectividades realizadas pelo GPS com outros satélites.

Figura 11.10 – Conectividade com satélites pelo GPS



Fonte: elaboração própria.

São necessários ao menos 4 satélites para que haja o enlace da comunicação, sendo possível notar pela figura que houve momentos no qual a conexão foi de 4 para nenhum (menos de 4 satélites conectados). A quantidade máxima de satélites simultâneos foi de 11 e durante a ascensão do balão e após a queda permaneceu próxima de 8 conexões.

Para concluir, o uso do BMP280 garante maior confiabilidade, tendo em vista a sensível conexão do GPS durante a missão, o que fez com que não funcionasse corretamente devido ao mal sinal ou algum problema com o módulo, este substituído para a 4ª fase.

São esperados na 4ª fase uma leitura mais consistente do GPS, tendo em vista a mudança da posição da antena, de modo que o sinal não tenha que penetrar a estrutura, o que pode enfraquecê-lo. O BMP280 foi substituído, pois parou de funcionar dias após a missão, este aparenta ter um funcionamento além do previsto no datasheet. Os termistores não irão realizar leituras abaixo de 0°C pois foram calibrados de 0-90°C. A estrutura foi projetada para manter o Cubesat dentro da faixa de operação de 0-70°C, refletindo parte da radiação térmica e mantendo aquecida a estrutura.

12. TESTES 4ª FASE

A etapa de testes do satélite partiu majoritariamente das simulações pelo método dos Elementos Finitos devido a indisponibilidade de alguns recursos para execução

prática dos testes.

12.1 Análise de força G:

Durante o lançamento do satélite são geradas forças devido a aceleração do veículo na subida, sendo esta a condição de contorno para a direção da força gravitacional, apontada para baixo (-Y), como visto nos resultados do apêndice K. Foi aplicado uma força máxima de 30G, ou seja

$F_G = 30.(9{,}80665\ m/s^2) \cong 294\text{N}.$

As demais definições aplicadas na simulação partem do engaste na base do satélite, representando a base do "dispenser" ou a superfície de outro Cubesat, condições de contato, propriedades do material (próprio da biblioteca do Autodesk Inventor). Espera-se, também, uma leve inclinação do veículo, contudo a resultante das forças continuará sendo maior de +Y a -Y (sistema de eixos do modelo).

Dado o exposto, os resultados mostram que devido as placas do circuito estarem presas aos espaçadores, os deslocamentos são maiores nas placas do circuito com no máximo 0,053mm no centro da placa, em direção de -Y. Com base no descrito, o deslocamento é baixo demais para afetar a integridade da estrutura.

12.2 Análise da Resposta em Frequência Modal:

A simulação a seguir, apresentada no apêndice L, se deve a análise das vibrações provenientes do lançamento do veículo lançador, podendo atingir frequências de até 233Hz.

Para a simulação é necessária a análise modal a fim de atribuir um valor de fator de amortecimento adequado para a estrutura, nesse caso foi atribuído um valor de aproximadamente 2%. Em sequência, o modelo de simulação parte para a Resposta em Frequência Modal, onde os parâmetros da simulação partem das condições do ambiente (força G e peso), disposição da estrutura e fator de amortecimento. Isto posto, foram adicionadas placas auxiliares de 100x100x2mm acima do Cubesat e na base, a fim de representar as faces de um ou mais Cubesats entre o ICARUS-1, ou o próprio dispenser.

Os parâmetros da força G são os mesmos da simulação anterior. Já o peso, parte da massa medida da estrutura final do satélite, esta de 523g, ou seja

$P = 9{,}80665.0{,}523 \cong 5{,}129\text{N}$

A direção da força peso (P) é para baixo (ou -Y), aplicada próximo ao centro de massa da estrutura sobre a placa de circuito central.

A condição de engaste se dá nas placas auxiliares e na região das faces laterais da estrutura, por estarem em contato com o dispenser.

É possível notar que o maior deslocamento visto na estrutura foi de 0,147mm. Muito baixo para as dimensões do Cubesat, o que o torna adequado para as vibrações sofridas durante o lançamento.

CONCLUSÃO

Os dados da 3ª fase mostraram que o Cubesat conseguiu realizar leituras suficientes para a missão empregada, apesar dos problemas durante lançamento e falha de comunicação com GPS. Os termistores realizaram leituras próximas às temperaturas medidas por BMP280 e MPU6050.

A pressão medida pelo BMP280 se aproxima, mesmo com desvio considerável, da pressão absoluta lida pelo GPS, o que demonstra a viabilidade de uso do GPS na ausência de sensores de pressão atmosférica.

A estrutura teve um isolamento adequado dado às temperaturas lidas e ausência de danos por umidade e baixas temperaturas, tendo em vista que choveu até o resgate da sonda e o satélite retornou ainda operacional, com 70% de bateria, aproximadamente.

Com base no descrito em relatório da 3ª fase e melhorias propostas para a 4ª fase. É esperado que o isolamento térmico mantenha o circuito em faixas operacionais de temperatura, e vida útil da bateria adequada até o resgate.

Foram vistos também baixos valores de deslocamento devido a vibrações de até 233Hz e força gravitacional de até 30G, o que torna a estrutura adequada para lançamento.

APÊNDICE A – Código do satélite, 4ª fase

```
// Equipe: ICARUS, código do satélite: ICARUS-1
// Autores: Adriano César de Sousa Pereira, Kayque Batista de Oliveira, João Pedro Bineli Alves, Kevin Cohin Hereda de Freitas Marinho
// Instituição responsável: IFSP - Campus São João da Boa Vista, UNESP - Campus São João da Boa Vista
// Código da 3ª fase disponível em: https://github.com/SirCesar/ICARUS-1/blob/main/Final_v17.2/Final_v17.2.ino
// Código da 4ª fase disponível em: https://github.com/SirCesar/ICARUS-1/blob/main/Final_v19/Final_v19.ino
//
#include <WiFi.h>             // biblioteca do módulo wi-fi
#include <HTTPClient.h>       // biblioteca de cliente http
#include <TinyGPSPlus.h>      // biblioteca do GPS
#include <Wire.h>             // biblioteca de endereçamento
#include <SPI.h>              // biblioteca de comunicação SPI
#include <Adafruit_BMP280.h>  // biblioteca do BMP pela Adafruit
#include <Adafruit_MPU6050.h> // Biblioteca do MPU6050 pela Adafruit
#include <Adafruit_Sensor.h>  // Biblioteca de complemento para MPU6050 pela Adafruit
#include <FS.h>               // biblioteca do leitor de cartão SD
#include <SD.h>               // biblioteca do leitor de cartão SD
//
// DADOS DE CONEXÃO WI-FI, LOCAL:
String serverName = "https://obsat.org.br/teste_post/envio.php";  // servidor de testes da olimpíada
char *ssid = "usuario";                                  // nome da rede wi-fi local
char *password = "senha";                                // senha da rede wi-fi local
String conteudo_WiFi = "";
int cont_conexao = 0;
// PÁGINA WEB COM RECEPÇÃO DOS DADOS ENVIADOS:
// https://obsat.org.br/teste_post/index.php                // use este link para ver os dados enviados
// DADOS DE CONEXÃO WI-FI, LOCAL DO EVENTO:
/*
String serverName = "http://192.168.0.1/";
char *ssid = "OBSAT_WIFI";
char *password = "OBSatZenith1000";
*/
// ----------------------------------------------------------------
// Tempo de leitura:
unsigned long t0 = 0;
unsigned long t1 = 0;
//
// INSTANCIANDO O PAYLOAD:
#define S0 32                                    // GPIO 32 (usando ADC)
#define S1 33                                    // GPIO 33 (usando ADC)
#define S2 25                                    // GPIO 25 (usando ADC)
#define S3 26                                    // GPIO 26 (usando ADC)
const int analog = 34;                             // sig = GPI O4
```

```
unsigned myAnalogRead(short inputCH, short an_in);         // função para leitura do MUX 16
#define canais 7                                          // número de pinos usados no MUX
int cont_canais = 0;                                       // contador de canais usados
float valor[canais];                                       // matriz com tamanho de canais, ou seja 7
const int pinos_canais[canais] = { 0, 1, 2, 3, 4, 5, 15 };  // divisor de tensão, pinos do mux
// Dados de calibração (cada item refere-se a um termistor):
const float A[canais] = { 7.080, 8.397, 5.565, 15.464, 5.969, 5.909 };
const float B[canais] = { -45.291, -57.141, -33.678, -120.117, -34.059, -34.854 };
const float C[canais] = { 104.323, 141.606, 73.709, 339.504, 68.471, 72.845 };
const float D[canais] = { -72.765, -121.328, -42.582, -370.290, -27.292, -33.766 };
const float E[canais] = { 11.649, 33.488, 2.146, 128.043, -4.785, -3.214 };
float valorT[6];       // valorT armazena leitura analógica de 0-6 termistores
double temp;           // variável com valores positivos e negativos
int nivelBateria = 0;  // armazena percentual de carga da bateria
// ---------------------------------------------------------------------
// INSTANCIANDO O MÓDULO SD:
String s_conteudo = "";
char const *char_conteudo = s_conteudo.c_str();
unsigned long contador_0 = 0;  // contador de escrita no cartão
// ---------------------------------------------------------------------
// INSTANCIANDO O ACELERÔMETRO/GIROSCÓPIO:
Adafruit_MPU6050 mpu;                  // renomeia a bliblioteca Adafruit_MPU6050 para mpu, a fim de simplificação quando chamada
float Gx, Gy, Gz, Ax, Ay, Az, mpu_temp;  // Eixos e temperatura
String mpu_gyro = "[0.00,0.00,0.00]", mpu_acel = "[0.00,0.00,0.00]";
// ---------------------------------------------------------------------
// INSTANCIANDO O BMP280:
#define BMP_SCK (13)               // pino SCK do módulo
#define BMP_MISO (12)              // pino MISO do módulo
#define BMP_MOSI (11)              // pino MOSI do módulo
#define BMP_CS (10)                // pino CS do módulo
Adafruit_BMP280 bmp;               // renomeia a biblioteca Adafruit_BMP280 para bmp, a fim de simplificação quando chamada
float bmp_temp, bmp_pres, bmp_alt;  // bmp_temp = temperatura do bmp280; bmp_pres = pressão do bmp280; bmp_alt = altitude do bmp280
// ---------------------------------------------------------------------
// INSTANCIANDO O GPS:
#define RXD2 16  // RXD2=GPIO16, RX do ESP32 com TX do GPS
#define TXD2 17  // TXD2=GPIO17, TX do ESP32 com RX do GPS
TinyGPSPlus gps;                                           // renomeia a biblioteca TinyGPSPlus para gps, a fim de simplificação quando chamada
float ALTURA_GPS = 0.0, LAT_GPS = 0.0, LGN_GPS = 0.0, KMPH_GPS = 0.0;  // LAT_GPS = Latitude do GPS; LGN_GPS = Longitude do GPS; KMPH = velocidade em km/h do GPS
int SAT_GPS = 0;                                           // SAT_GPS = quantidade de satélites conectados ao módulo GPS, a fim de triangulação do sinal (mínimo 4)
float gps_pres = 0.0;                                      // variável que irá armazenar o cálculo de pressão com altura do GPS (ALTURA_GPS)
unsigned long intervalo_gps = 1000;
//
```

```
// Instanciando LoRa ED32-433T20DC:
#define RXD1 13  // RXD1=GPIO13, RX do ESP32 com TX do ED32
#define TXD1 27  // TXD1=GPIO27, TX do ESP32 com RX do ED32
String conteudo_LoRa = "0";
// ----------------------------------------------------------------
void setup() {
  // Setup aqui
  // ----------------------------------------------------------------
  Serial.begin(9600);  // velocidade de transmissão para serial, taxa de 9600 baud
  // ----------------------------------------------------------------
  // Inicializa ED32-477T20DC em UART 1:
  Serial1.begin(9600, SERIAL_8N1, RXD1, TXD1);
  // ----------------------------------------------------------------
  // Inicializa GPS em UART 2:
  Serial2.begin(9600, SERIAL_8N1, RXD2, TXD2);
  // Inicializa núcleo 2:
  xTaskCreatePinnedToCore(loop2, "loop2", 10000, NULL, 1, NULL, 1);
  // ----------------------------------------------------------------
  // Conexão Wi-Fi:
  WiFi.begin(ssid, password);              // inicializa wi-fi com nome e senha da rede local
  while (WiFi.status() != WL_CONNECTED && cont_conexao < 200) {  // se o status de
conexão do wi-fi é de "conectado" e tempo passado <= 9800ms, executa o comando
    delay(50);
    cont_conexao += 1;
    Serial.print(".");
    if (WiFi.status() == WL_CONNECTED) {
      Serial.println("\nWi-Fi conectado!");
    }
    Serial.println("\nWi-Fi não conectado!");
  }
  // ----------------------------------------------------------------
  // Cartão SD e arquivo:
  SD.begin();
  writeFile(SD, "/Teste_1.txt", "\n");
  escreve_string("n° escrita, bmp_temp, bmp_pres, bmp_alt, TT101, TT102, TT103, TT104,
TT105, TT106, nivelBateria, Gx, Gy, Gz, Ax, Ay, Az, mpu_temp, LAT_GPS, LGN_GPS,
KMPH_GPS, SAT_GPS, ALTURA_GPS, gps_pres");
  // ----------------------------------------------------------------
  // Multiplexador:
  pinMode(S0, OUTPUT);  // define Sn, como saída, onde n vai de 0-3, como consta na
inicialização
  pinMode(S1, OUTPUT);
  pinMode(S2, OUTPUT);
  pinMode(S3, OUTPUT);
  setMux(0);
  // ----------------------------------------------------------------
  // SENSOR MPU6050:
  mpu.begin();                             // inicializa o girosccópio/acelerômettro
  mpu.setAccelerometerRange(MPU6050_RANGE_8_G);  // define faixa de medição do
acelerômetro
```

```
  mpu.setGyroRange(MPU6050_RANGE_500_DEG);       // define faixa de medição do
giroscópio
  mpu.setFilterBandwidth(MPU6050_BAND_5_HZ);     // define precisão da leitura
  //---------------------------------------------------------------
  // SENSOR BMP280:
  bmp.begin(0x76);  // inicializa o sensor bmp280
  // Dados padrões de configuração do datasheet:
  bmp.setSampling(Adafruit_BMP280::MODE_NORMAL,      // Modo de operação
                  Adafruit_BMP280::SAMPLING_X2,      // Sobreamostragem de temperatura
                  Adafruit_BMP280::SAMPLING_X16,     // Sobreamostragem de pressão
                  Adafruit_BMP280::FILTER_X16,       // Filtragem
                  Adafruit_BMP280::STANDBY_MS_500);  // Período de descanço (standby)
  // ---------------------------------------------------------------
}
// ---------------------------------------------------------------------
void writeFile(fs::FS &fs, const char *path, const char *message) {
  //Serial.printf("Escrito para o arquivo: %s\n", path);
  File file = fs.open(path, FILE_WRITE);  // abre o arquivo para escrita, path = arquivo;
FILE_WRITE = conteúdo a ser escrito

  if (!file) {
    //Serial.println("Falhou em abrir arquivo para escrita");
    //return;
  }
  if (file.print(message)) {
    //Serial.println("Arquivo escrito");
  } else {
    //Serial.println("Falhou em escrever");
  }

  file.close();  // fecha arquivo após escrita
}
void appendFile(fs::FS &fs, const char *path, const char *message) {
  //Serial.printf("Atribuido para o arquivo: %s\n", path);
  File file = fs.open(path, FILE_APPEND);  // adiciona conteúdo ao arquivo criado em
writeFile, path = arquivo; FILE_APPEND = conteúdo a ser atribuído;

  if (!file) {
    //Serial.println("Falha em abrir arquivo para atribuição");
    //return;
  }
  if (file.print(message)) {
    //Serial.println("Mensagem atribuída");
  } else {
    //Serial.println("Falha na atribuição");
  }

  file.close();
}
// ----------------------------FUNÇÕES GERAIS-------------------------------
```

```
void mpu_6050() {
  sensors_event_t a, g, temp;   // quando ocorre variação em a, aceleração, g, giro e t, temperratura
  mpu.getEvent(&a, &g, &temp);  // atribui os valores de aceleração, giro ee temperatura em a, g e temp
  // Leitura acelerômetro (m/s²):
  Ax = a.acceleration.x;
  Ay = a.acceleration.y;
  Az = a.acceleration.z;
  // Leigura giroscópio (°/s):
  Gx = g.gyro.x;
  Gy = g.gyro.y;
  Gz = g.gyro.z;
  mpu_temp = temp.temperature;
  mpu_gyro = "[" + String(Gx, 2) + "," + String(Gy, 2) + "," + String(Gz, 2) + "]";  // unidade em LSB/(°/s)
  mpu_acel = "[" + String(Ax, 2) + "," + String(Ay, 2) + "," + String(Az, 2) + "]";  // unidade em LSB/(g/s)
}
void pressao_calc(float h_) {  // recebe altura do GPS
  double p = (101325) * pow((1 - (2.25577E-5) * h_), 5.25588);
  gps_pres = (float)p;  // converte de double para float
  // Fonte da equação: https://www.engineeringtoolbox.com/air-altitude-pressure-d_462.html
}
void dados_gps() {
  if (gps.location.isValid() == 1) {     // se há dados coletados, atribui nas variáveis globais
    LAT_GPS = gps.location.lat();        // latitude
    LGN_GPS = gps.location.lng();        // longitude
    KMPH_GPS = gps.speed.kmph();         // velocidade
    SAT_GPS = gps.satellites.value();    // satélites conectados
    ALTURA_GPS = gps.altitude.meters();  // altura
  } else {                               // se não há dados, mantém valores zerados como float
    LAT_GPS = 0.0;
    LGN_GPS = 0.0;
    KMPH_GPS = 0.0;
    SAT_GPS = 0;
    ALTURA_GPS = 0.0;
  }
}
String payload() {
  String dados = "[" + String(LAT_GPS, 2) + "," + String(LGN_GPS, 2) + "," + String(KMPH_GPS, 2) + "," + String(SAT_GPS) + "," + String(ALTURA_GPS, 2) + "," + String(gps_pres, 2) + "," + String(nivelBateria) + "," + String(bmp_alt, 2) + "," + String(bmp_temp, 2) + "," + String(mpu_temp, 2) + "," + String(valorT[0], 2) + "," + String(valorT[1], 2) + "," + String(valorT[2], 2) + "," + String(valorT[3], 2) + "," + String(valorT[4], 2) + "," + String(valorT[5], 2) + "]";
  return dados;
  // A formatação segue a do servidor, formato JSON
  // Todos os valores são convertidos para String, em especial float recebe formatação com duas casas decimais, ex: String(numero, 2)
```

```
}
void leitura_bmp() {
  bmp_pres = bmp.readPressure();        // faz leitura de pressão
  bmp_temp = bmp.readTemperature();     // faz leitura de temperatura
  bmp_alt = bmp.readAltitude(1013.25);  // faz leitura de altitude usando como referência
1013.25 m (nível do mar)
}
void leituraMux() {
  float valorM = 0;                                                    // média dos valorres
inicializa em 0
  for (int j = 0; j < canais; j++) {                                    // laço de tamanho 0-7
    setMux(pinos_canais[j]);                                            // atribui cada valor em
pinos_canais[]
    for (int i = 0; i < 5; i++) {                                       // itera 5 vezes
      valorM += (float)0.9792 * (floatMap(analogRead(analog), 0.0, 4095.0, 0.0, 3.3)) +
0.4302;  // para cada iteração é feita leitura analógica
      delay(50);                                                        // atraso para o ADC ler com
sucesso
    }                                                                   // reinicia o laço para cada pino 5
vezes
    valorM = (float)valorM / 5.0;                                       // faz a média das 5
leituras
    valor[j] = valorM;                                                  // armazena o valor da
iteração atual em valor[j]
    valorM = 0;                                                         // zera valorM para a
próxima iteração de j
  }
  for (int i = 0; i < canais - 1; i++) {  //converte valores de tensão em Temperatura
    valorT[i] = A[i] * (pow(valor[i], 4.0)) + B[i] * (pow(valor[i], 3.0)) + C[i] * (pow(valor[i],
2.0)) + D[i] * valor[i] + E[i];
    // Atribui curva de calibração para cada termistor, dado uma leitura analógica
  }
  float tensao_b = valor[6] * 3.0;                          //leitura analógica da bateria, divisor de
tensão
  nivelBateria = floatMap(tensao_b, 3 * 2, 4.2 * 2, 0.0, 100.0);  //converte em porcentagem o
valor lido na bateria
  if (nivelBateria < 0) {                                   // ajuste para evitar erro de syntaxe
    //Serial.print("Erro de leitura na bateria (valor < 0): "); Serial.println(nivelBateria);
    nivelBateria = 0;
  } else if (nivelBateria > 100) {  // ajuste para evitar erro de syntaxe
    //Serial.print("Erro de leitura na bateria (valor > 0): "); Serial.println(nivelBateria);
    nivelBateria = 0;
  }
}
void setMux(short inputCH) {
  switch (inputCH) {
    case 0:  // combinação binária para leitura dos pinos de 0-6
      digitalWrite(S0, LOW);
      digitalWrite(S1, LOW);
      digitalWrite(S2, LOW);
```

```
      digitalWrite(S3, LOW);
      break;
    case 1:
      digitalWrite(S0, HIGH);
      digitalWrite(S1, LOW);
      digitalWrite(S2, LOW);
      digitalWrite(S3, LOW);
      break;
    case 2:
      digitalWrite(S0, LOW);
      digitalWrite(S1, HIGH);
      digitalWrite(S2, LOW);
      digitalWrite(S3, LOW);
      break;
    case 3:
      digitalWrite(S0, HIGH);
      digitalWrite(S1, HIGH);
      digitalWrite(S2, LOW);
      digitalWrite(S3, LOW);
      break;
    case 4:
      digitalWrite(S0, LOW);
      digitalWrite(S1, LOW);
      digitalWrite(S2, HIGH);
      digitalWrite(S3, LOW);
      break;
    case 5:
      digitalWrite(S0, HIGH);
      digitalWrite(S1, LOW);
      digitalWrite(S2, HIGH);
      digitalWrite(S3, LOW);
      break;
    case 6:
      digitalWrite(S0, LOW);
      digitalWrite(S1, HIGH);
      digitalWrite(S2, HIGH);
      digitalWrite(S3, LOW);
      break;
    case 15:
      digitalWrite(S0, HIGH);
      digitalWrite(S1, HIGH);
      digitalWrite(S2, HIGH);
      digitalWrite(S3, HIGH);
      break;
  }
}
float floatMap(float x, float in_min, float in_max, float out_min, float out_max) {
  return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}
String dados(int _e, int _b, float _t, float _p, String G, String A, String PL) {
```

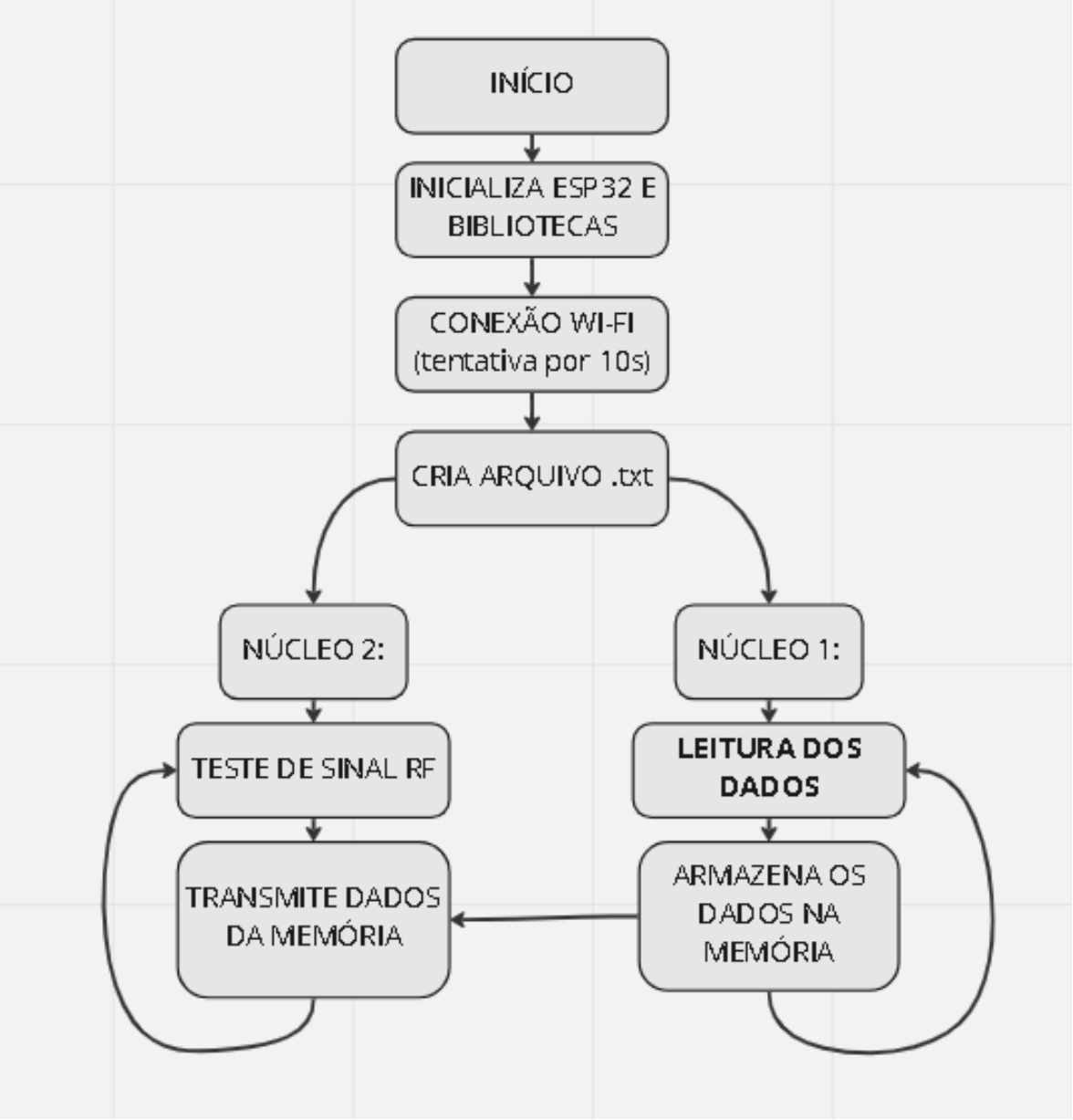
```
  // equipe, lv bateria, temperatura, pressão, giroscópio, acelerômetro, payload (termistor).
  String dados = " {\"equipe\":" + String(_e) + ",\"bateria\":" + String(_b) +
",\"temperatura\":" + String(_t, 2) + ",\"pressao\":" + String(_p, 2) + ",\"giroscopio\":" + G +
",\"acelerometro\":" + A + ",\"payload\":" + PL + "}";
  return dados;
  // São os dados enviados, também concatena o payload
}
void conexao(int estado) {                    // estado 0 = bmp, estado 1 = gps e mpu_temp
  HTTPClient http;                            // renomeia biblioteca HTTPClient para http
  http.begin(serverName.c_str());             // inicializa comunicação http com endereço
http definido
  http.addHeader("Content-Type", "application/json");  // define o formato da mensagem como
JSON
  if (estado == 1) {
    conteudo_WiFi = dados(50, nivelBateria, mpu_temp, gps_pres, mpu_gyro, mpu_acel,
payload());  // temperatura do mpu6050 e pressão calculada com altura do gps
    if (Serial.available()) {
      Serial.println(conteudo_WiFi);
    }
  } else if (estado == 0) {
    conteudo_WiFi = dados(50, nivelBateria, bmp_temp, bmp_pres, mpu_gyro, mpu_acel,
payload());  // temperatura do bmp280 com pressão própria do bmp280
    if (Serial.available()) {
      Serial.println(conteudo_WiFi);
    }
  }
  int httpResponse = http.POST(conteudo_WiFi.c_str());  // tipo de envio, POST, cujo
conteúdo é a variável toSend de string para char
  http.end();                                 // encerra comunicação http
}
void escreve_string(String _conteudo) {
  char_conteudo = _conteudo.c_str();          // recebe conteúdo em string e converte para
char
  appendFile(SD, "/Teste_1.txt", char_conteudo);  // atribui no arquivo o conteúdo em char
  appendFile(SD, "/Teste_1.txt", ",");        // virgula separador de dado
}
void escreve_int(int _conteudo) {
  s_conteudo = String(_conteudo);             // recebe conteúdo em inteiro e converte para
string
  char_conteudo = s_conteudo.c_str();         // recebe conteúdo em string e converte para
char
  appendFile(SD, "/Teste_1.txt", char_conteudo);  // atribui no arquivo o conteúdo em char
  appendFile(SD, "/Teste_1.txt", ",");
}
void escreve_float(float _conteudo) {
  s_conteudo = String(_conteudo, 2);          // recebe conteúdo em float e converte para float
com 2 casas decimais
  char_conteudo = s_conteudo.c_str();         // recebe conteúdo em string e converte para
char
  appendFile(SD, "/Teste_1.txt", char_conteudo);  // atribui no arquivo o conteúdo em char
```

```
  appendFile(SD, "/Teste_1.txt", ",");
}
// ---------------------------------------------------------------------
void loop() {
  t0 = millis();
  for (unsigned long start = millis(); millis() - start < intervalo_gps;) {
    while (Serial2.available()) {
      gps.encode(Serial2.read());  // faz a requisição durante 1000 ms, enviando vários pedidos
    }
  }
  // --- ETAPA DE LEITURA ---
  dados_gps();              // Dados do GPS: ALTURA_GPS, LAT_GPS, LGN_GPS,
KMPH_GPS, SAT_GPS
  pressao_calc(ALTURA_GPS);  // Calcula pressão com "ALTURA_GPS": gps_pres
  leitura_bmp();            // Dados do bmp: bmp_pres, bmp_alt, bmp_temp
  leituraMux();             // Termistor e nível de bateria: valorT[j], nivelBateria
  mpu_6050();               // Giroscópio: Gx, Gy, Gz, Ax, Ay, Az, mpu_gyro, mpu_acel
  // --- ETAPA DE ESCRITA ---
  appendFile(SD, "/Teste_1.txt", "\nEscrita: ");  // Nomeia escrita de cada dado, usando a
contagem como referência
  contador_0 += 1;                          // Enumera (itera) a contagem de escritas no cartão SD
  escreve_string(String(contador_0));
  // Dados do BMP280:
  escreve_float(bmp_temp);  // escreve dado em float para string (olhar função
escreve_float()), temperatura bmp280
  escreve_float(bmp_pres);  // pressão bmp280
  escreve_float(bmp_alt);   // altura bmp280
  // Dados do MUX:
  escreve_float(valorT[0]);   // temperatura termistor TT101
  escreve_float(valorT[1]);   // temperatura termistor TT102
  escreve_float(valorT[2]);   // temperatura termistor TT103
  escreve_float(valorT[3]);   // temperatura termistor TT104
  escreve_float(valorT[4]);   // temperatura termistor TT105
  escreve_float(valorT[5]);   // temperatura termistor TT106
  escreve_int(nivelBateria);  // escreve dado em int para string (olhar função escreve_int()),
percentual da bateria
  // Dados do giroscópio/acelerômetro:
  escreve_float(Gx);        // eixo X giroscópio em °/s
  escreve_float(Gy);        // eixo Y giroscópio em °/s
  escreve_float(Gz);        // eixo Z giroscópio em °/s
  escreve_float(Ax);        // eixo X acelerômetro em m/s²
  escreve_float(Ay);        // eixo Y acelerômetro em m/s²
  escreve_float(Az);        // eixo Z acelerômetro em m/s²
  escreve_float(mpu_temp);  // temperatura do MPU6050 em °C
  // Dados do GPS:
  escreve_float(LAT_GPS);     // latitude do GPS
  escreve_float(LGN_GPS);     // longitude do GPS
  escreve_float(KMPH_GPS);    // velocidade do GPS em km/h
  escreve_int(SAT_GPS);       // satélites conectados
  escreve_float(ALTURA_GPS);  // altura do GPS em metros (m)
```

```
  escreve_float(gps_pres);    // pressão calculada com altura do GPS em Pascal (Pa)
  t1 = millis() - t0;
  // --- ETAPA DE ESCRITA NA SERIAL ---
  Serial.println("Temperatura (bmp280): " + String(bmp_temp, 2));
  Serial.println("Pressão (bmp280): " + String(bmp_pres, 2));
  Serial.println("Altitude (bmp280): " + String(bmp_alt, 2));
  Serial.println("TT101: " + String(valorT[0], 2));
  Serial.println("TT102: " + String(valorT[1], 2));
  Serial.println("TT103: " + String(valorT[2], 2));
  Serial.println("TT104: " + String(valorT[3], 2));
  Serial.println("TT105: " + String(valorT[4], 2));
  Serial.println("TT106: " + String(valorT[5], 2));
  Serial.println("Bateria: " + String(nivelBateria));
  Serial.println("Giroscópio (mpu6050): " + mpu_gyro);
  Serial.println("Acelerômetro (mpu6050): " + mpu_acel);
  Serial.println("Temperatura (mpu6050): " + String(mpu_temp, 2));
  Serial.println("Latitude (gps): " + String(LAT_GPS, 2));
  Serial.println("Longitude (gps): " + String(LGN_GPS, 2));
  Serial.println("Velocidade (gps): " + String(KMPH_GPS, 2));
  Serial.println("Satélites conectados (gps): " + String(SAT_GPS));
  Serial.println("Altitude (gps): " + String(ALTURA_GPS));
  Serial.println("Pressão (gps): " + String(gps_pres, 2));
}
void loop2(void *parameter) {
  for(;;)
  {
    // Envia os dados por Wi-Fi ou LoRa:
    if (ALTURA_GPS < 8900.00) {  // aguarda o tempo descorrido que foi calculado
anteriormente                                          // grava o tempo de início do envio dos dados
    // é feita a conexão com as variáveis globais dos dados lidos nos sensores, esse processo
dura 1,1-1,2 segundos
      if(WiFi.status() == WL_CONNECTED)
      {
        conexao(0);
      }
      conteudo_LoRa = dados(50, nivelBateria, bmp_temp, bmp_pres, mpu_gyro, mpu_acel,
payload());
      Serial1.print(conteudo_LoRa);
    }
    if (ALTURA_GPS > 8900.00) {  // se o wi-fi está conectado e a altura for superior a
8900.00 m
      if (WiFi.status() == WL_CONNECTED)
      {
        conexao(1);
      }
      conteudo_LoRa = dados(50, nivelBateria, mpu_temp, gps_pres, mpu_gyro, mpu_acel,
payload());
      Serial1.print(conteudo_LoRa);
    }
    delay(t1);
```

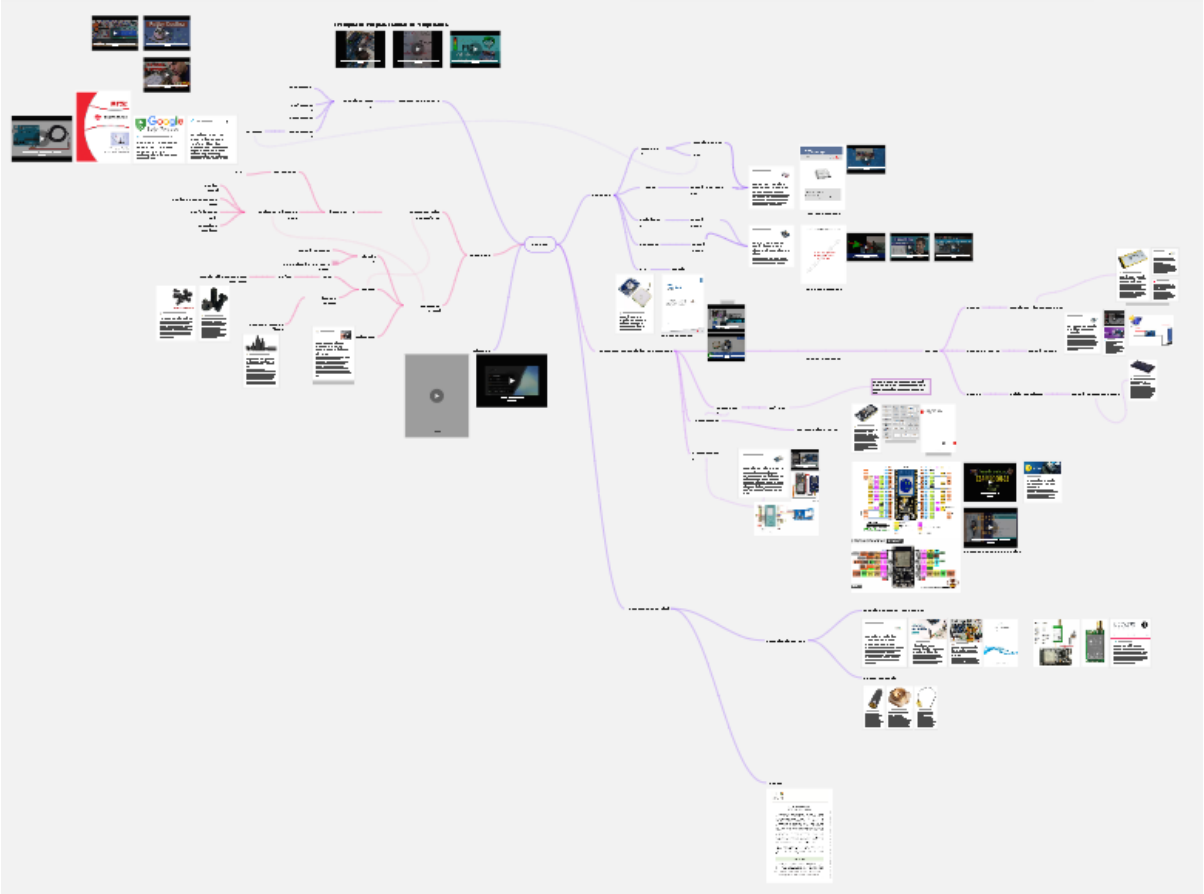
```
  }
  vTaskDelete(NULL);
}
```

APÊNDICE B – Fluxograma de Código

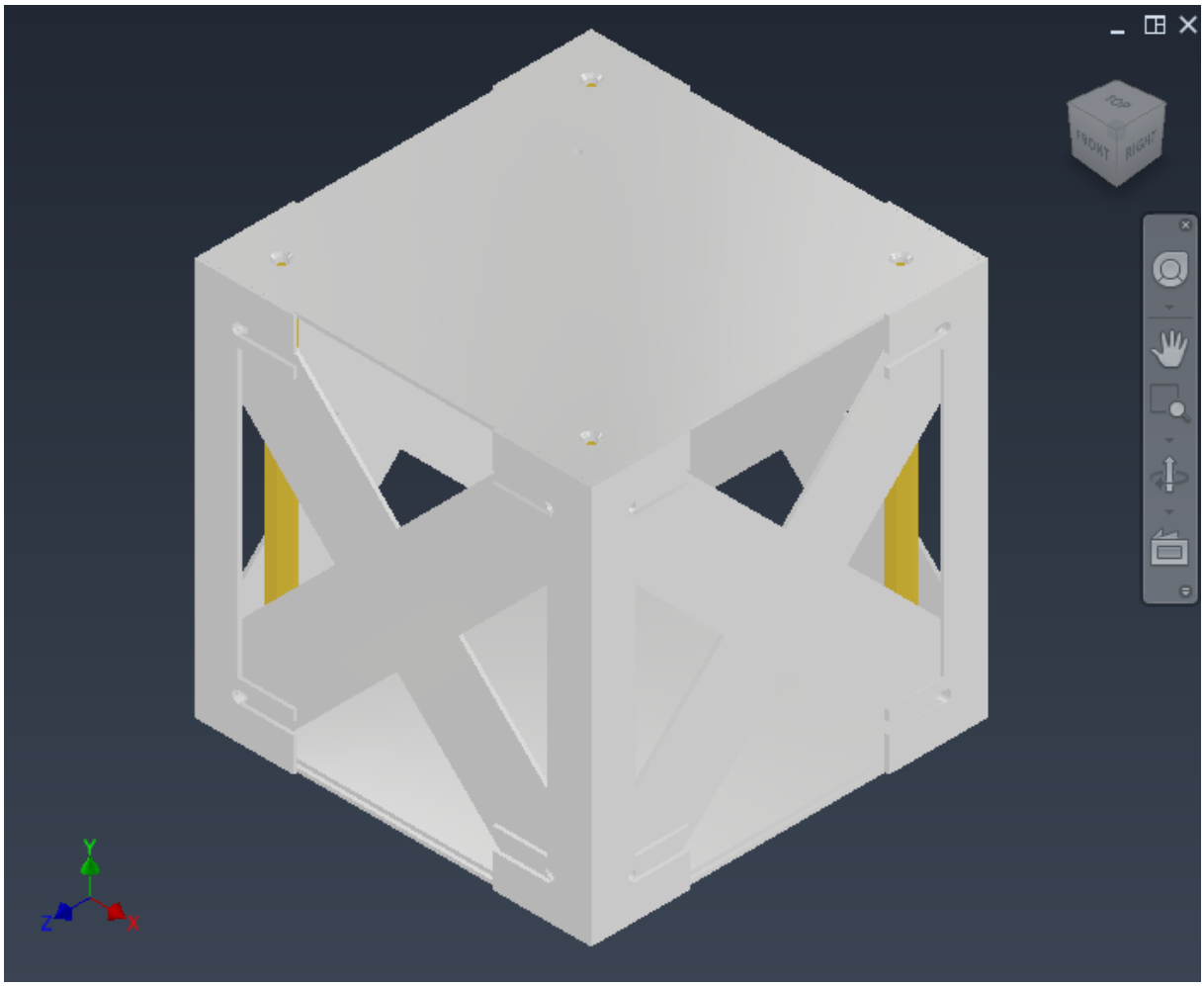


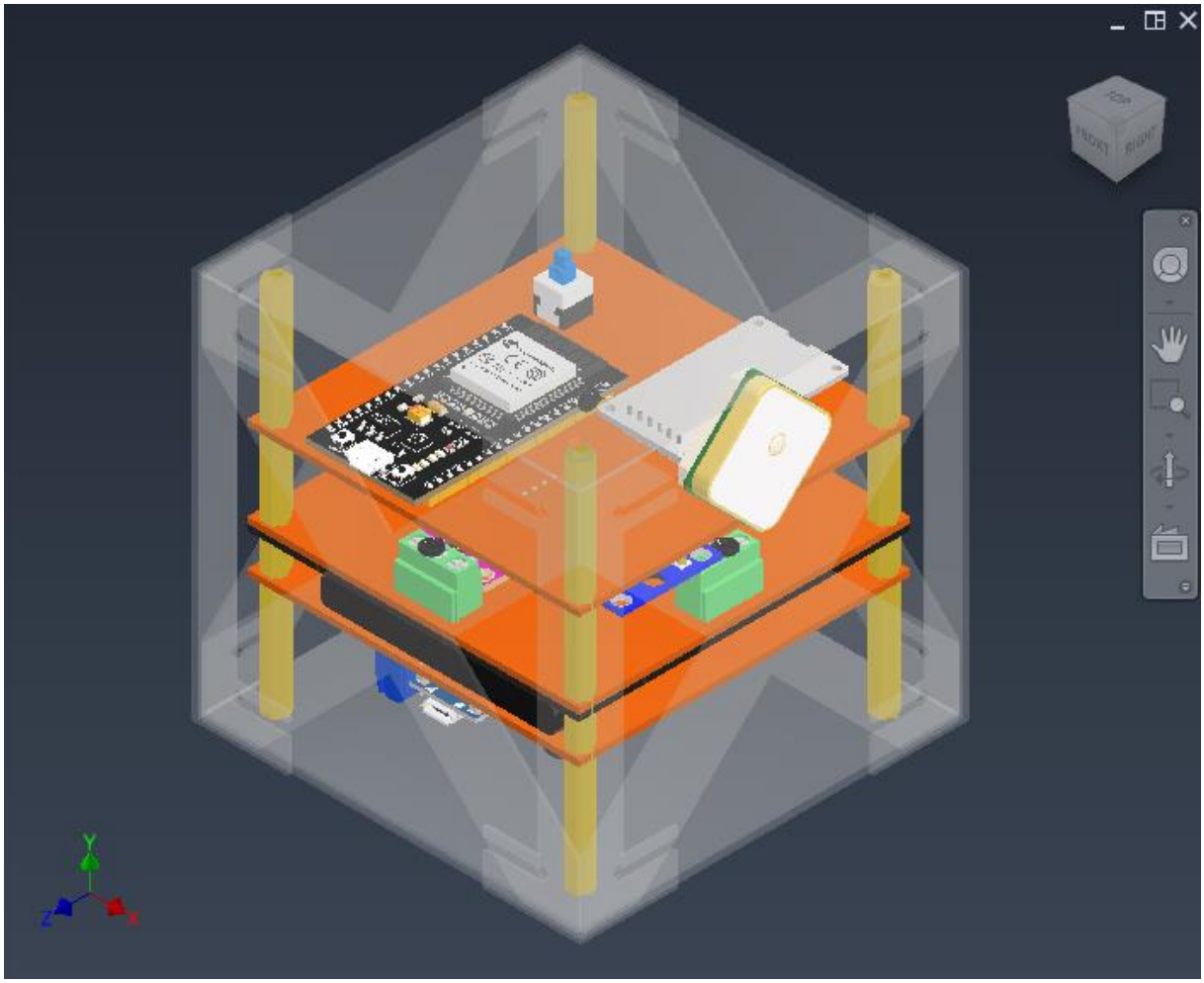
APÊNDICE C – Link de acesso aos dados coletados, 3ª fase
https://github.com/SirCesar/ICARUS-1/blob/main/Dados_3a_fase.txt

APÊNDICE D – Plano de projeto por mapa conceitual

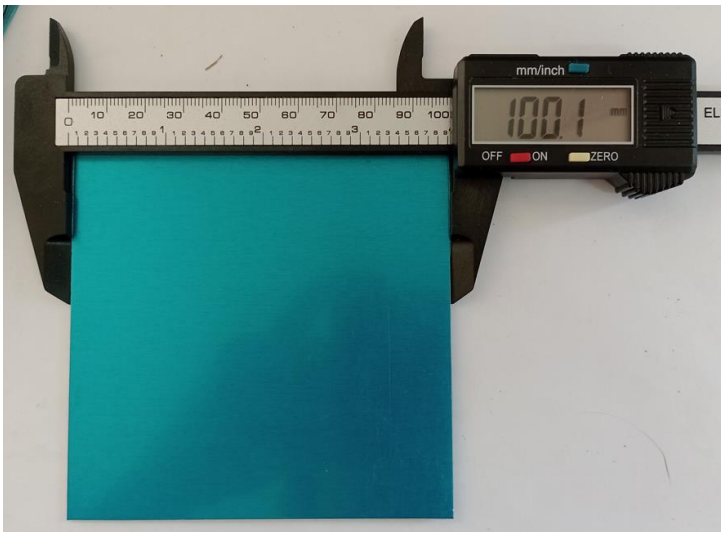


APÊNDICE E – Modelagem 3D

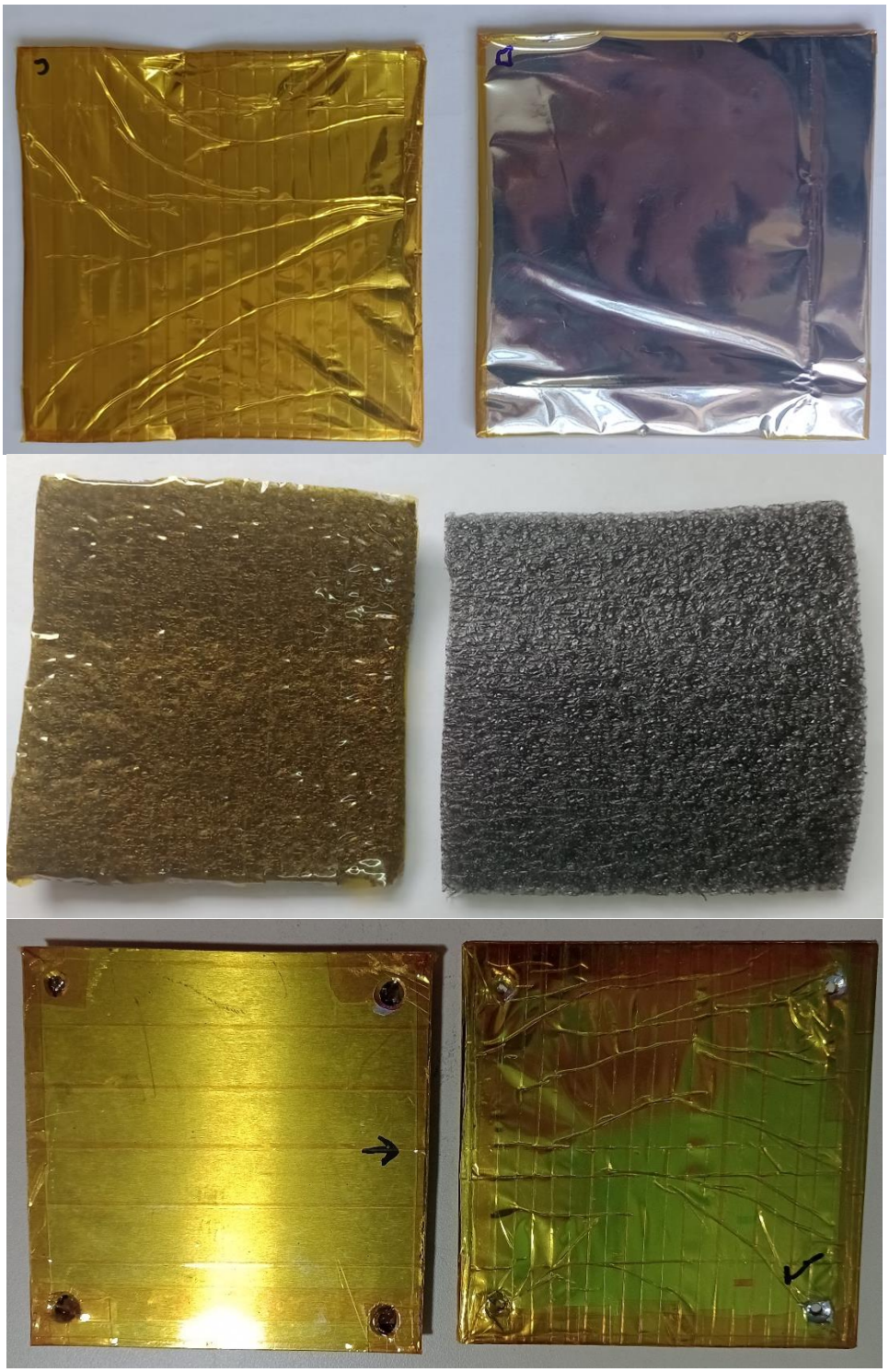


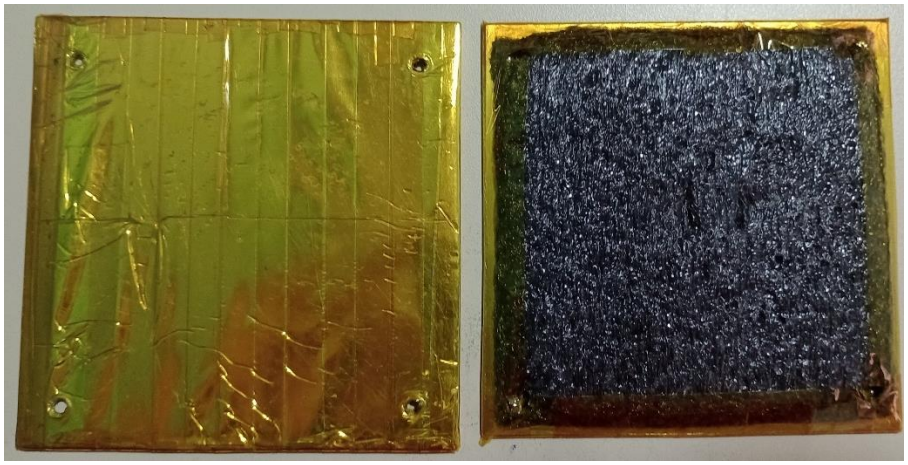


APÊNDICE F – Manufatura da estrutura



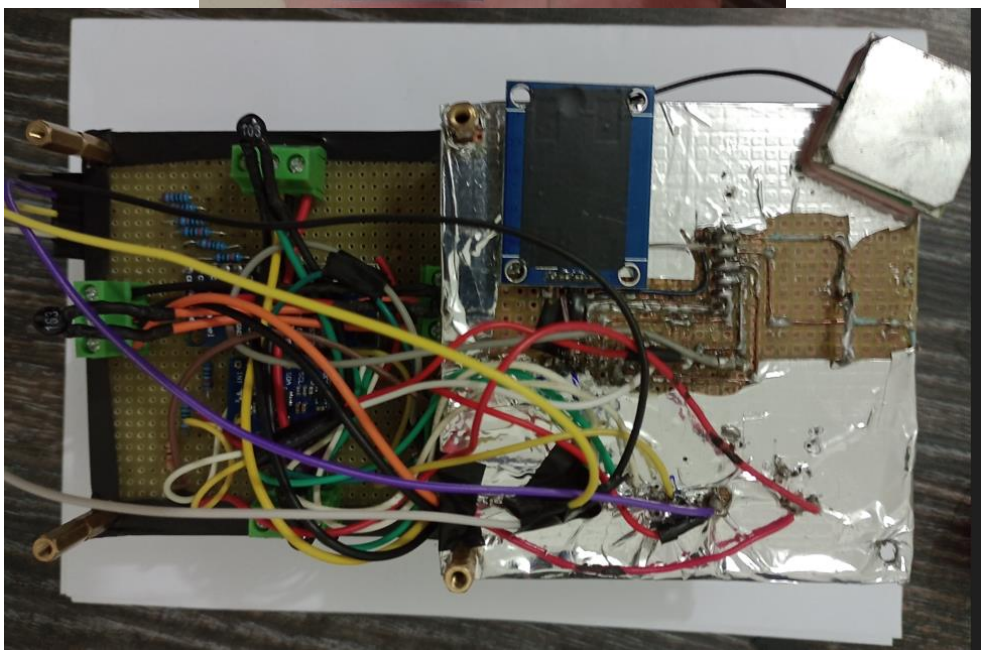
APÊNDICE G – Manufatura do isolamento térmico

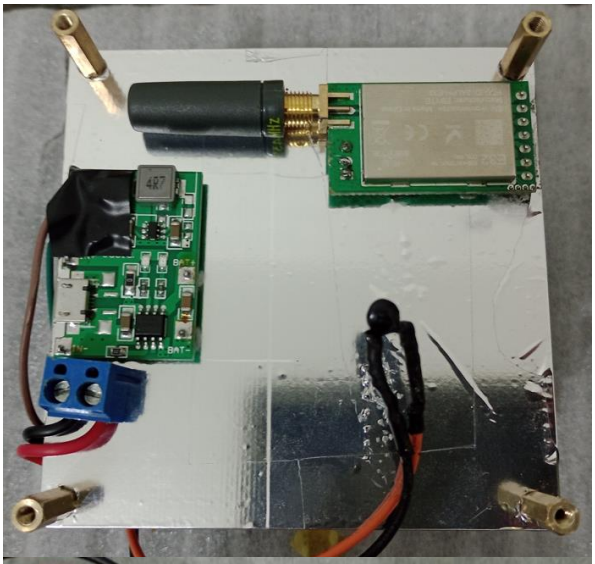


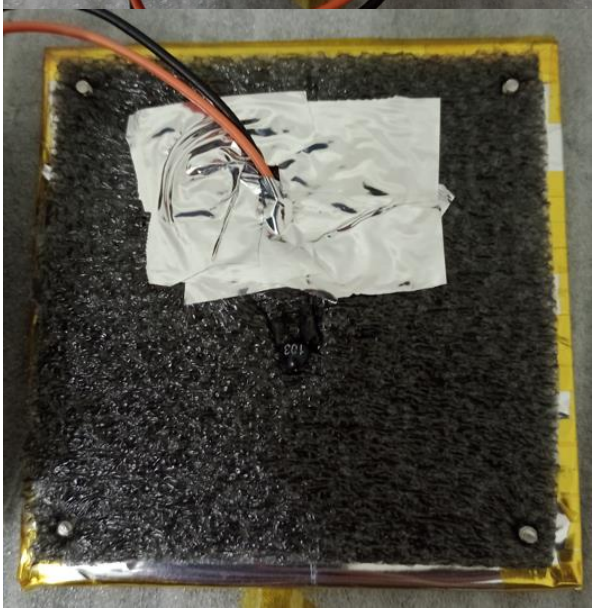


APÊNDICE H – Estrutura completa

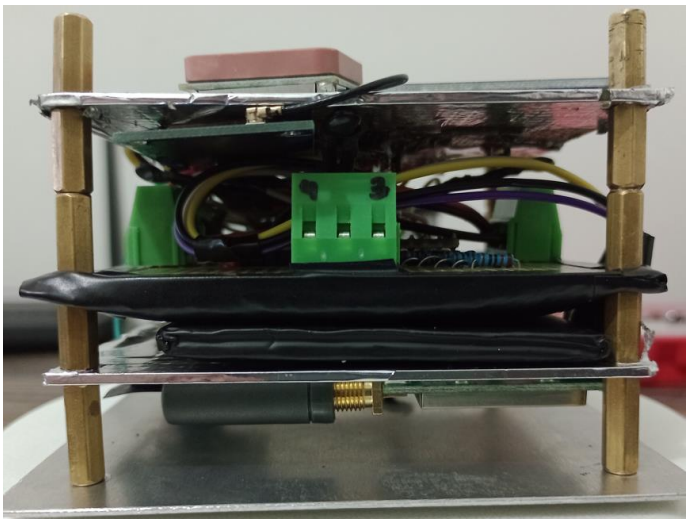
APÊNDICE I – Placas de circuito




4R7
BAT+
BAT-

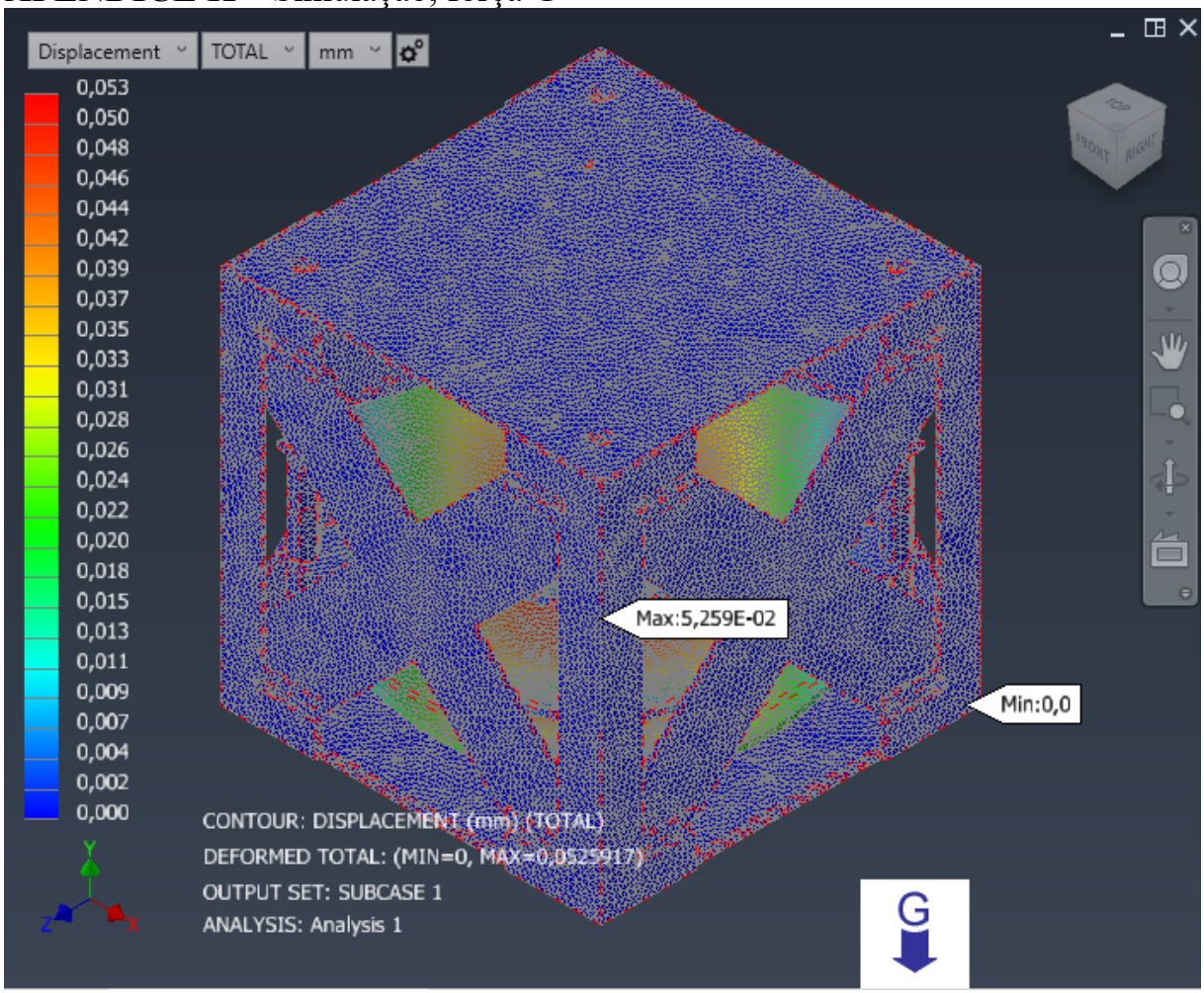

103

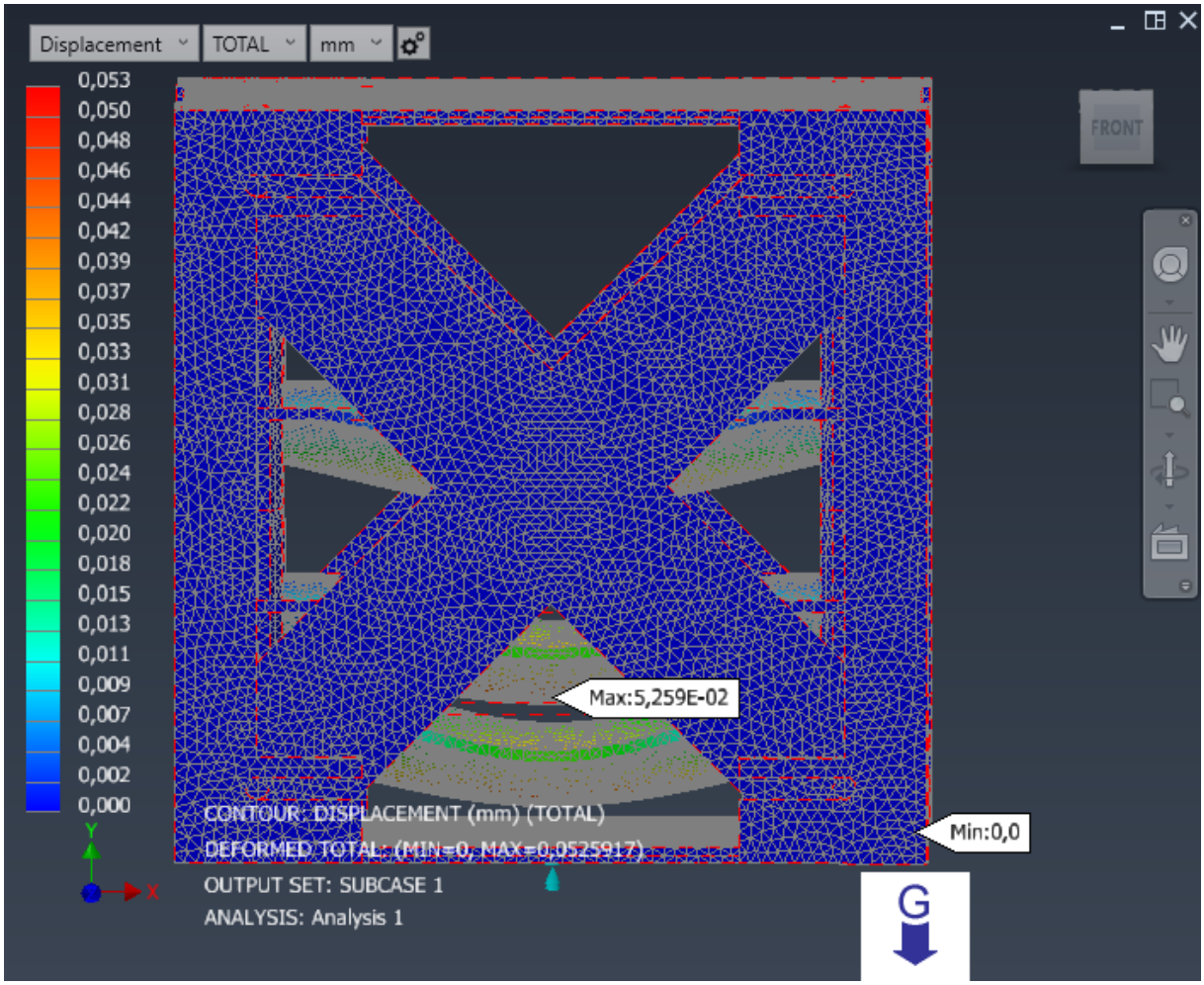
APÊNDICE J – Subsistema de potência



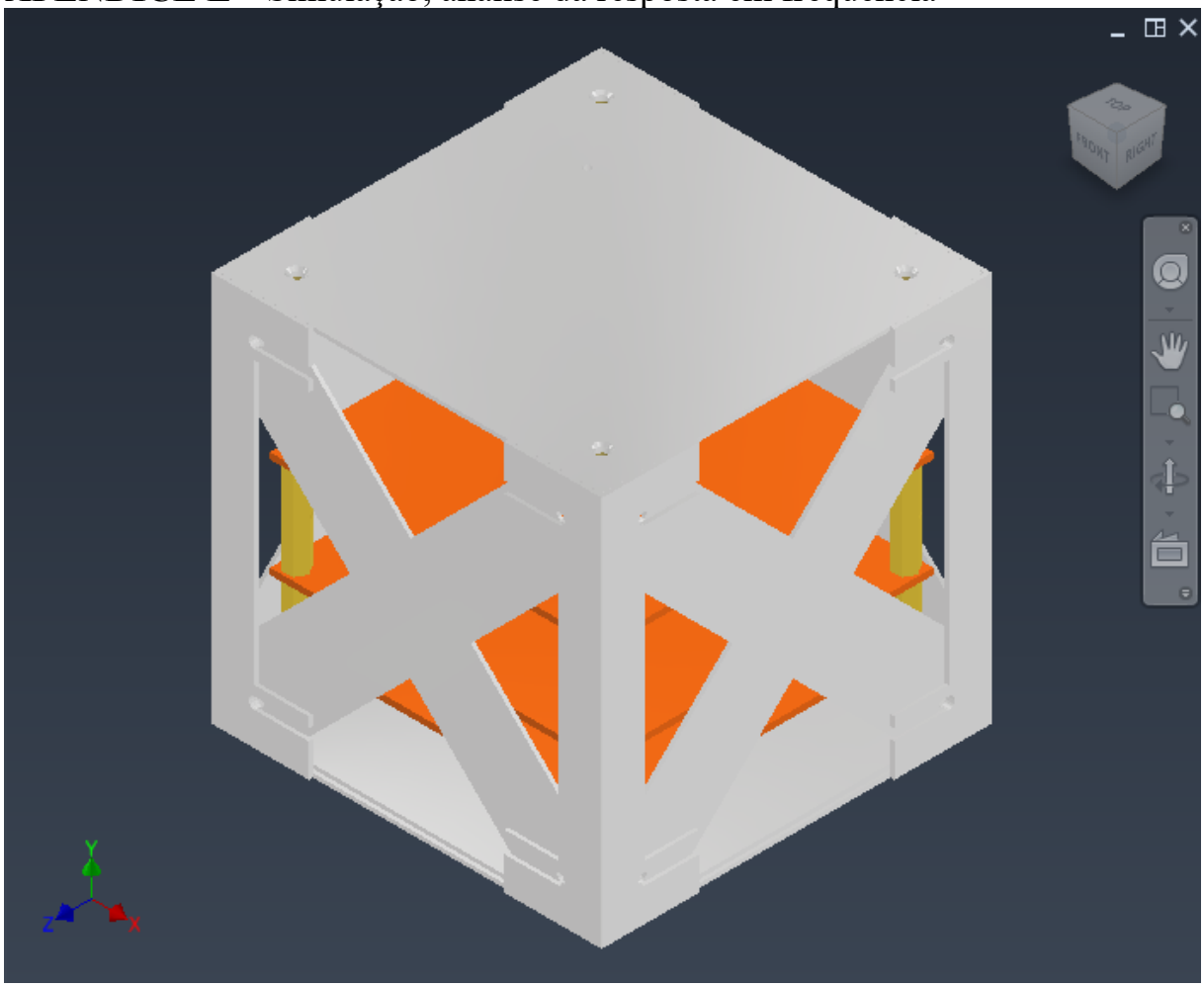

+ Sony W
US3865854H5
- 0237KM045Y002A

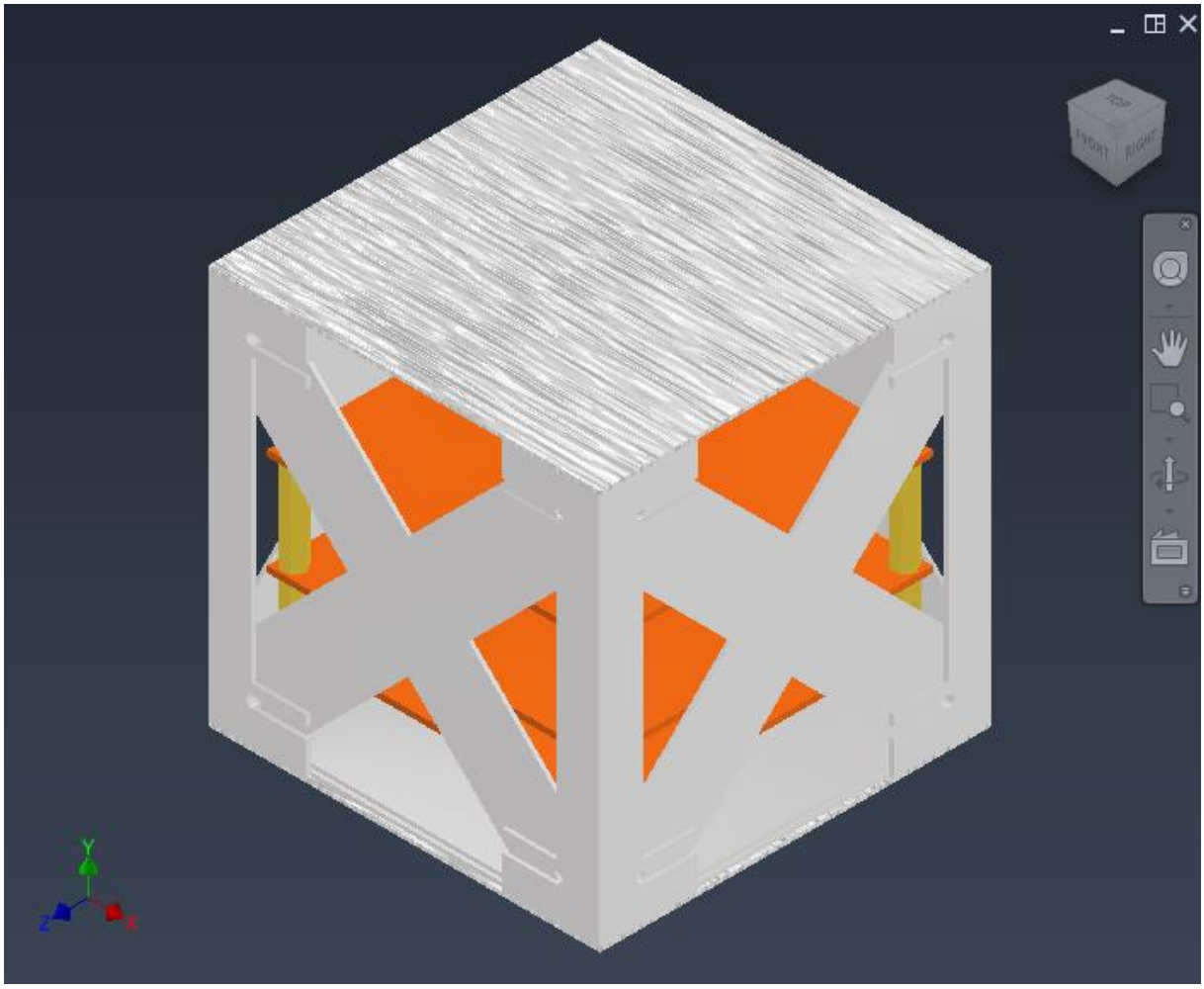
APÊNDICE K – Simulação, força G

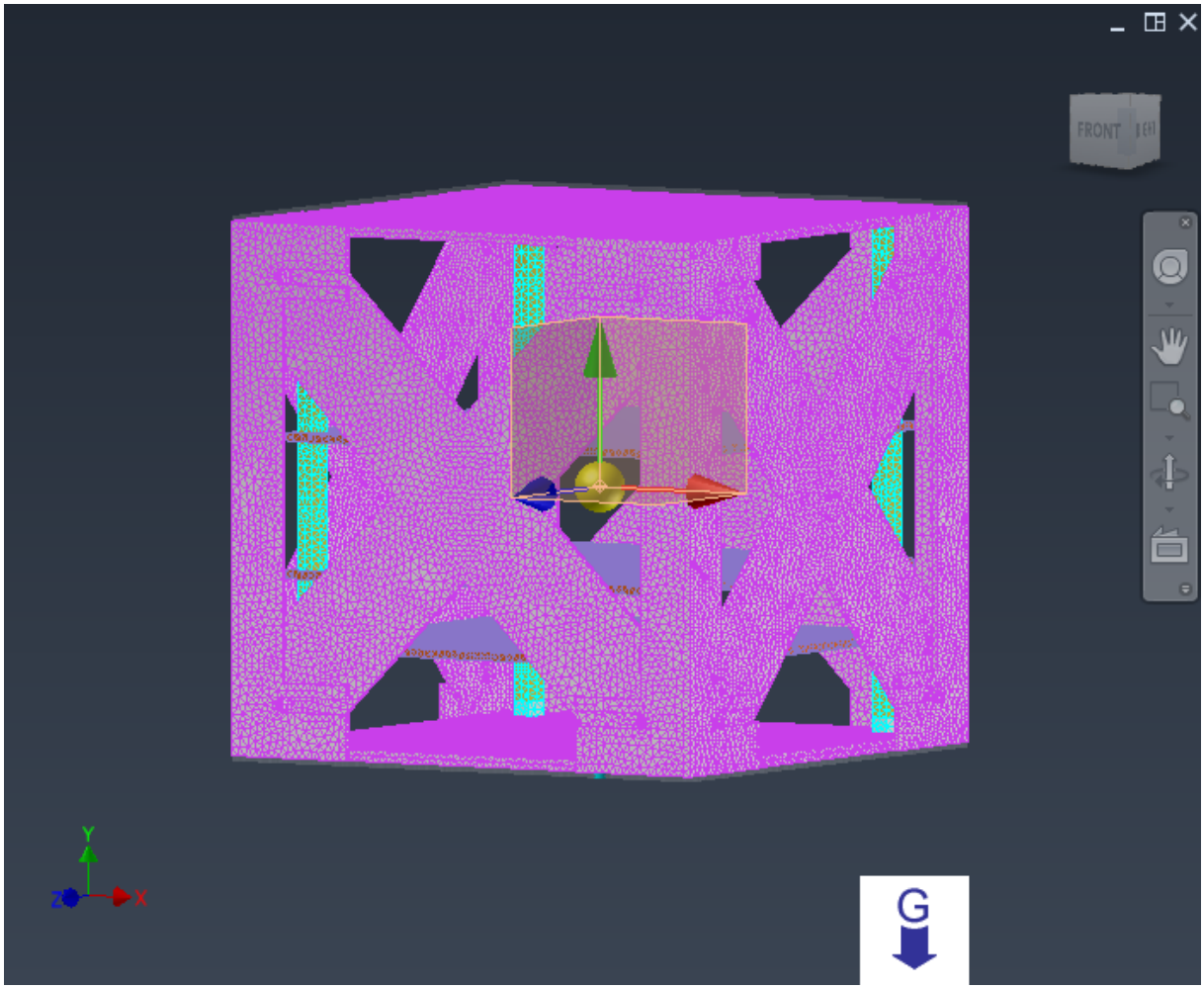


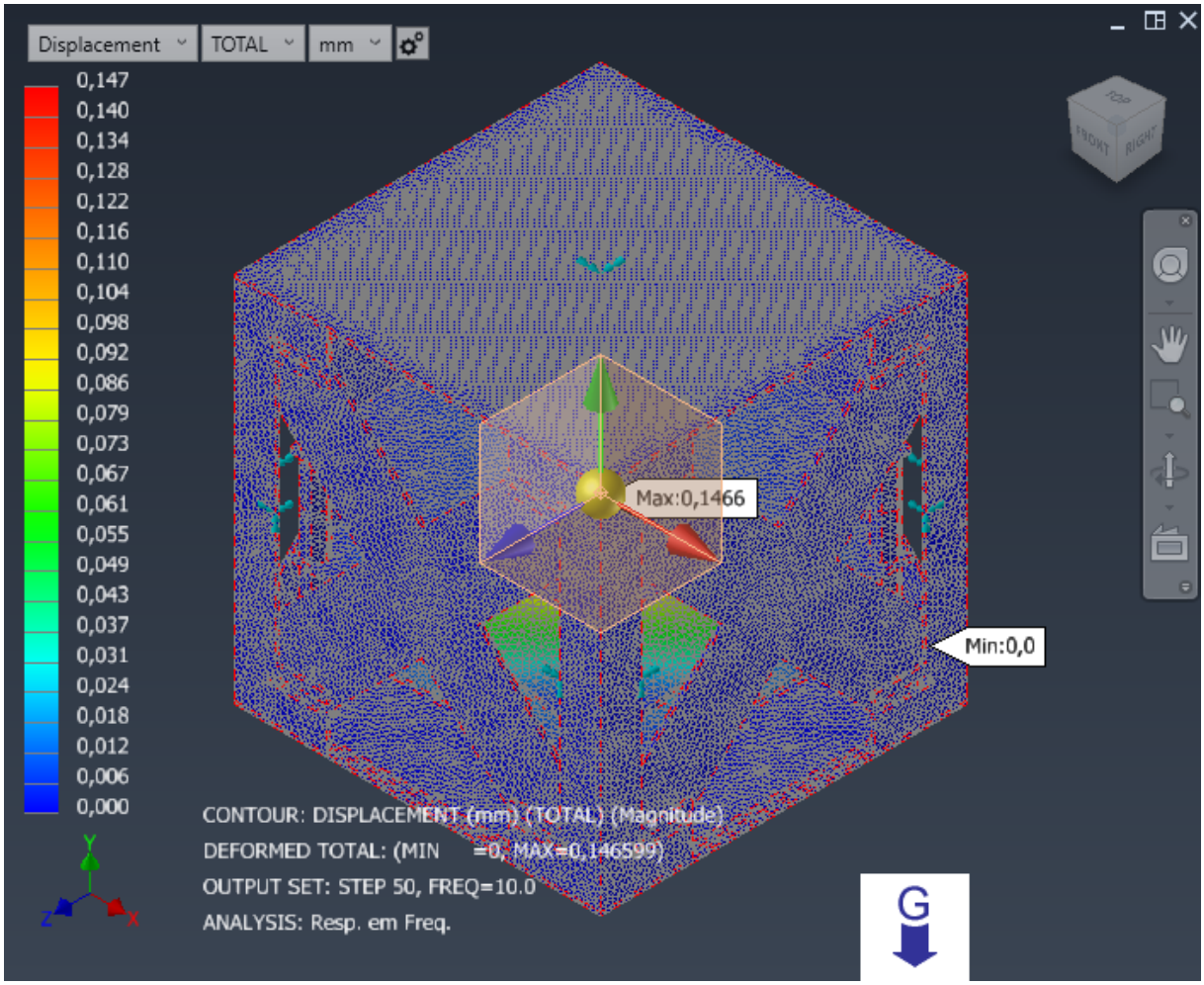

Displacement
TOTAL
mm
0,053
0,050
0,048
0,046
0,044
0,042
0,039
0,037
0,035
0,033
0,031
0,028
0,026
0,024
0,022
0,020
0,018
0,015
0,013
0,011
0,009
0,007
0,004
0,002
0,000
FRONT
Max:5,259E-02
Min:0,0
CONTOUR: DISPLACEMENT (mm) (TOTAL)
DEFORMED TOTAL: (MIN=0; MAX=0,0525917)
OUTPUT SET: SUBCASE 1
ANALYSIS: Analysis 1
Y
X
G

APÊNDICE L – Simulação, análise da resposta em frequência




Y
Z
X


FRONT
Y
Z
X
G


Displacement
TOTAL
mm
0,147
0,140
0,134
0,128
0,122
0,116
0,110
0,104
0,098
0,092
0,086
0,079
0,073
0,067
0,061
0,055
0,049
0,043
0,037
0,031
0,024
0,018
0,012
0,006
0,000
Max:0,1466
Min:0,0
CONTOUR: DISPLACEMENT (mm) (TOTAL) (Magnitude)
DEFORMED TOTAL: (MIN =0, MAX=0,146599)
OUTPUT SET: STEP 50, FREQ=10.0
ANALYSIS: Resp. em Freq.
Y
Z
X
G