TRABALHO DE GRADUAÇÃO

ESTUDO DE DESEMPENHO DE SISTEMA EM TEMPO REAL PARA SIMULADOR DE PEQUENOS SATÉLITES

Lukas Lorenz de Andrade

Brasília, Maio de 2021

ENGENHARIA
MECATRÔNICA
UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia
Curso de Graduação em Engenharia de Controle e Automação

TRABALHO DE GRADUAÇÃO

ESTUDO DE DESEMPENHO DE SISTEMA EM TEMPO REAL PARA SIMULADOR DE PEQUENOS SATÉLITES

Lukas Lorenz de Andrade

Relatório submetido como requisito parcial de obtenção de grau de Engenheiro de Controle e Automação

Banca Examinadora

Prof. Dr. Renato Alves Borges, ENE/UnB _____
Orientador

Prof. Dr. Daniel Chaves Café, ENE/UnB _____
Examinador interno

Prof. Dr. João José Costa Gondim, CIC/UnB _____
Examinador interno

Brasília, Maio de 2021

FICHA CATALOGRÁFICA

LUKAS, LORENZ DE ANDRADE
ESTUDO DE DESEMPENHO DE SISTEMA EM TEMPO REAL PARA SIMULADOR DE PEQUENOS SATÉLITES

[Distrito Federal] 2021.

x, 101p., 297 mm (FT/UnB, Engenheiro, Controle e Automação, 2021). Trabalho de Graduação – Universidade de Brasília. Faculdade de Tecnologia.

1. Simulador de Pequenos Satélites
2. Sistema em Tempo Real
3. Internet das Coisas
4. Unidade de Medição Inercial
5. Visão Computacional

I. Mecatrônica/FT/UnB
II. Título (Série)

REFERÊNCIA BIBLIOGRÁFICA

ANDRADE, LUKAS LORENZ DE, (2021). ESTUDO DE DESEMPENHO DE SISTEMA EM TEMPO REAL PARA SIMULADOR DE PEQUENOS SATÉLITES. Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT.TG-$n°$022, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 101p.

CESSÃO DE DIREITOS

AUTOR: Lukas Lorenz de Andrade

TÍTULO DO TRABALHO DE GRADUAÇÃO: ESTUDO DE DESEMPENHO DE SISTEMA EM TEMPO REAL PARA SIMULADOR DE PEQUENOS SATÉLITES

GRAU: Engenheiro ANO: 2021

É concedida à Universidade de Brasília permissão para reproduzir cópias deste Trabalho de Graduação e para emprestar ou vender tais cópias somente para propósitos acadêmicos e científicos. O autor reserva outros direitos de publicação e nenhuma parte desse Trabalho de Graduação pode ser reproduzida sem autorização por escrito do autor.

Lukas Lorenz de Andrade

Núcleo Rural Boa Esperança II, Ch 1130, Lago Norte.

71507-991 Brasília – DF – Brasil.

Dedicatória

Dedico este trabalho à minha mãe, Gardy Lorenz, cujo amor e força me permitiram sonhar e realizar os meus sonhos.

Lukas Lorenz de Andrade

Agradecimentos

Agradeço primeiramente aos meus pais, Gardy e Narciso, que forneceram toda a base e condições para a realização dessa etapa na minha vida. As minhas irmãs que me ajudaram e me apoiaram durante essa trajetória. Agradeço aos meus amigos Flávio, João, Mateus e Carol que me ajudaram a realizar esse trabalho, de forma indireta, e no apoio nos momentos de dificuldade e indecisões. Agradeço ao meu orientador, Renato, pela oportunidade, atenção e carinho, ajudando a lapidar esse trabalho. Agradeço a todos os integrantes do laboratório LODESTAR pelas contribuições que propiciaram esse trabalho, como o Rodrigo, Letícia, Igor e Lucas. Agradeço a todos os outros amigos e colegas que não foram citados mas que contribuiram em momentos críticos da minha formação.

Lukas Lorenz de Andrade

RESUMO

Dada a dificuldade de testes em ambiente espacial, foi criado no Laboratório de Simulação e Controle de Sistemas Aeroespaciais (LODESTAR) Universidade de Brasília (UnB) um simulador capaz de reproduzir o campo magnético e baixo atrito semelhantes a condições de órbita por meio de uma mesa instrumentada sobre um rolamento a ar centrada na gaiola de Helmholtz. Assim, propicia o desenvolvimento de tecnologias e aplicação direta de conhecimento científico em projetos de nanosatélites, principalmente, no padrão cubesat com baixíssimo custo.

No entanto, para a implementação de algoritmos de controle, estabilização (B-dot) ou determinação de atitude, necessita-se de um aparato tecnológico capaz de garantir determinada confiabilidade nos dados computados e sensores, além de ser modular, podendo com pequenos esforços trocar dispositivos de comunicação ou atuação, fatores que o simulador atual necessitava ser melhorado. Vale ressaltar que a análise de erro dos sensores envolvidos no processo foi realizada, mas não havia a preocupação com os atrasos do sistema e em redundâncias dos sensores, de forma que a confiabilidade dos dados torna-se questionável sob a ótica de sistemas em tempo real.

Este trabalho consiste no prosseguimento da reformulação do simulador de pequenos satélites, de forma a se transformar o simulador em sistema em tempo real, análise de confiabilidade dos dados da IMU e do ADCV (*Attitude Determination by Computer Vision*) e dos atrasos ocasionados pelo protocolo de comunicação entre a estação solo e a mesa instrumentada, o qual será o Wifi (IEEE 802.11) nos experimentos realizados em laboratório.

Palavras Chave: Simulador de Pequenos Satélites, Sistema em Tempo Real, Internet das Coisas, Unidade de Medição Inercial, Visão Computacional.

ABSTRACT

Due to the difficult found in test nanosatellites in space environment, the Aerospace system Simulation and Control Laboratory (LODESTAR) University of Brasilia (UnB) was founded to replicate the magnetic field and frictionless conditions found in such places. The micro gravity conditions is emulated by a test-bed which is composed by an air bearing table centered in the Helmholtz cage. Thereby, it provides the technology and scientific knowledge development and application in cubesats nanosatellites projects with low cost.

Accuracy and precision are essential not only in the control algorithms, stabilization (B-dot) or attitude determination, but the measured data and it is important that simulator can be modular in a manner that it is able to change easily the control and communication algorithms or the actuator and radio devices. The actual state of the test-bed simulator does not correspond to that expectations. It is important to say that the sensors errors was analysed but a system error and delay was not computed so far. So in real time systems (RTS) the performance and accuracy of the system is questionable.

This work is a continuation of previous academic publications seeking out to transform the test-bed simulator in a real time platform. Therefore it will be proposed a scientific methodology to infer the probabilistic worst case response time (pWCRT) which is the principal metric of a RTS, measure the inertial measurement unity (IMU) and the attitude determination by computer vision subsystem (ADCV) accuracy and precision and the wifi (IEEE 802.11 protocol) delay which is the communication protocol used in the laboratory experiments.

Keywords: Small Satellite Simulator, Real Time System, Internet of Things, Inertial Measurement Unit, Computer Vision.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

Símbolos Gregos

ω	Velocidade angular	[rad/s]
ϕ	Ângulo de rolagem (X)	
θ	Ângulo de guinada (Y)	
ψ	Ângulo de arfagem (Z)	

Subscritos

ref	referência
lb	comunicação em *loopback*
Z	referente a componente Z do sistema de coordenadas
Y	referente a componente Y do sistema de coordenadas
X	referente a componente X do sistema de coordenadas
$2d$	sistema de referência em 2 dimenssões, geralmente o da câmera
$3d$	sistema de referência em 3 dimenssões, geralmente o do mundo físico
m	massa do sistema

Sobrescritos

$\cdot$	Variação temporal (derivada temporal)
$-$	Valor médio

Siglas

J	Momento de inercia ou matriz do momento de inércia
t	Definição de Tempo Contínuo
s	Notação para seno
c	Notação para cosseno
r	Braço de alavanca
m	Massa do sistema
g	Gravidade
TH	Limiar utilizado no POT
CPU	Unidade Central de Processamento
AO	Saída Analógica - *Analog Out*
DO	Saída Digital - *Digital Out*
SDA	Pino de dado da comunicação i2c - *Serial Data*
SCL	Pino de clock da comunicação i2c - *Serial Clock*
RST	Reiniciar - *Reset*
CS	Seletor de *Chip* - *Chip Select*
OBC	Computador embarcado - *Onboard Computer*
IMU	Unidade de Media Inercial - *Inertial Measurement Unit*
ADCV	Sistema de detemrinação de atitude por visão computacional - *Attitude Determination by Computer Vision*
RT	Tempo Real - *Real Time*
RTOS	Sistema Operacional em Tempo Real - *Real Time Operating System*
WCET	Tempo de Execução no Pior Caso - *Worst Case Execution Time*
WCRT	Tempo de Resposta no Pior Caso - *Worst Case Response Time*
pWCRT	Tempo de Resposta Probabilístico no Pior Caso - *Probabilistic Worst Case Response Time*
ESC	Flag que Sinaliza a Interrupção de Saída do Sistema
SUDOESC	Flag que Sinaliza o Reinicio do Sistema
UPPAAL	Software Utilizado para Validação Lógica da Modelagem do Sistema em Tempo Real

Capítulo 1

Introdução

1.1 Contextualização

O Laboratório de Simulação e Controle de Sistemas Aeroespaciais (LODESTAR) vem participando de diversas iniciativas espaciais como a missão Alfa Crux e a Wormsail. Em experiências anteriores, pode-se destacar as missões Kuaray e Serpens. As missões consistem no lançamento de nanossatélites no padrão Cubesat, onde cada unidade U corresponde a um cubo de dimenssões 10x10x10 centímetros.

Com o crescente investimento no setor aeroespacial desde a última década, criou-se o padrão Cubesat cujo objetivo é tornar o espaço mais acessível devido aos componentes utilizados e a impressão 3D qualificada para voo. Dessa forma, o investimento em softwares de simulação, simuladores e em pesquisa cresceu proporcionalmente, de modo que se tornou evidente a necessidade de um simulador que emulasse as condições de órbita com maior confiabilidade e utilizando-se de componentes de hardware que permitissem maior modularização de todos os seus componentes.

O simulador de pequenos satélites [2, 3, 5, 8] permite que algoritmos de controle ou estabilização (*detumbling*) como o B-dot [9, 10] possa ser validado e com pequenos ajustes reutilizado para missões reais como as citadas anteriormente. O simulador consiste em uma mesa instrumentada apoiada sobre um mancal a ar e centrada em um gaiola de Helmholtz. A mesa contém massas móveis que são capazes de anular o torque gravitacional, ou seja, os efeitos da gravidade sobre a rotação da mesma, e o campo magnético gerado pelas bobinas anulam o campo terrestre e geram a componente vetorial do campo no ponto de órbita emulado.

Para tanto, sobre a mesa intrumentada é colocado o Cubesat ou o Mockup, réplica a qual não é qualificada para vôo e serve para validação dos conceitos aplicados antes da missão. A mesa possui 3 rodas de reação que são motores DC com massas acopladas no terminal de seu eixo de transmissão, de forma a propiciar rotação com 3 graus de liberdade. Além desse tipo de atuador, tem-se o magnetorque, cujo principio de funcionamento é baseado na variação do campo magnético produzido por suas bobinas, gerando movimento. A respeito da dinâmica da mesa, pode-se simplificar como um pêndulo invertido, vide Figura 1.1, favorecido pelo mancal a ar que produz um atrito viscoso residual que pode ser desprezível.

Figura 1.1: Figura representativa da mesa instrumentada sobre o mancal a ar com o Mockup 2U sobre. Fonte: referência [1].

Vale ressaltar que a dinâmica e os efeitos considerados serão expostos no Capítulo 2.

1.2 Simulador de Pequenos Satélites

O simulador de pequenos satélites é responsável por emular condições de órbita. Assim, como na referência [11], percebe-se que mesmo para o padrão guarda chuva, em que a mesa de testes fica no centro da gaiola de helmholtz, há diferentes tipos de topologias e especificações.

Vale ressaltar que o simulador de pequenos satélites é o resultado de diversos projetos realizados no laboratório LODESTAR - Universidade de Brasília. Assim, a primeira contribuição para a gaiola de helmholtz se deu nas referências [8, 12]; a mesa instrumentada nas referências [2, 3, 13, 14]; o algoritmo de *detumbling* com a primeira versão do ADCV em uma câmera [9, 10, 5]; e a reformulação do simulador com a expansão do ADCV para 3 eixos [1] com a nova arquitetura proposta de hardware e software [15, 4] e contribuições especiais para a comunicação TCP [6, 16]. Dessa forma, esse trabalho compreende o projeto e a implementação de um sistema em tempo real para o ADCV e a sincronização das câmeras entre si e com as as outras funcionalidades do sistema. Assim como a validação desse projeto.

1.2.1 Hardware e Mecânica do Simulador

Pode-se dividir o simulador de pequenos satélites do LODESTAR em três principais subsistemas: gaiola de helmholtz, mesa instrumentada e a unidade de processamento (estação solo), como apresentado na Figura 1.9. No entanto, neste trabalho o foco se dará na mesa instrumentada (*Test-bed*) e na unidade de processamento, que são representados pelo OBC, ou a central de processamento, e o ADCV, respectivamente.

1.2.1.1 Gaiola de Helmholtz

A gaiola de helmholtz consiste em um par de bobinas alinhadas para cada eixo, de forma que se tem um campo magnético uniforme no ponto médio entre as bobinas, ou seja, no centro da gaiola, quando é considerado os 3 eixos. Assim, para cada bobina se tem uma fonte do modelo Agilent 6032A DC capaz de prover 60V ou 50A em 1000W, conforme apontado na referência [8], indicado na Figura 1.2. Além disso, foi utilizado cerca de 800m de cobre AWG14 por par de bobina com diâmetro de 1.628mm, corrente nominal máxima de 5.9A e resistência de 8.282Ω/Km, resultando e uma resistência aferida de 6.90Ω, 6.84Ω e 5.57Ω para as bobinas externas, do meio e internas.

Figura 1.2: Figura representativa da gaiola de helmholtz e do mancal a ar alimentado pelo compressor.

No processo de geração de campo pelas fontes, há a compensação do campo magnético da Terra a partir da calibração do magnetômetro HMR2300 que gera um offset em cada componente produzido pela bobina correspondente [12]. A partir do propagador orbital, um script em matlab

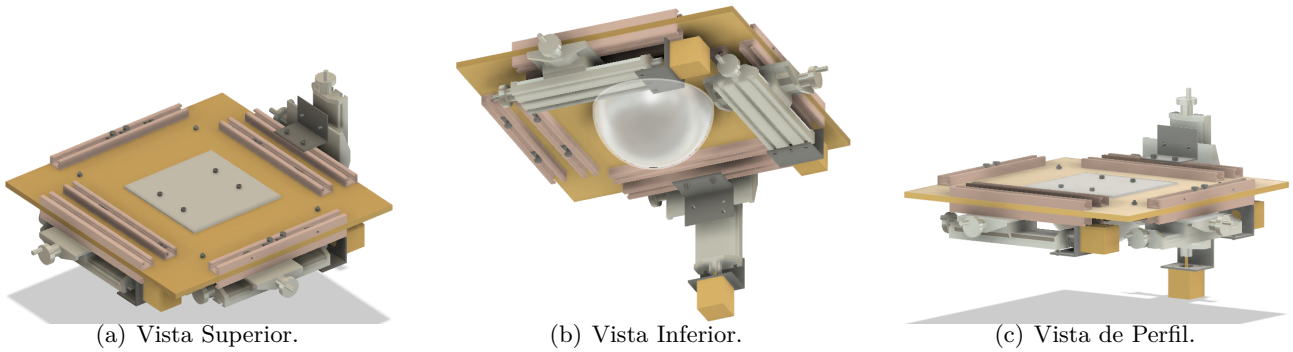
que gera valores de corrente para as fontes a partir do ponto de órbita simulado, emula-se o campo correspondente no simulador seguindo as limitações previamente mencionadas de corrente do cobre AWG14.

No propagador orbital [12] ainda tem uma malha de controle que usa o magnetômetro como feedback para a geração das componentes do campo magnético naquele ponto de órbita. Dessa forma, pode-se considerar uma resultante de campo constante até mesmo para o magnetômetro (atuador magnético), já que a dinâmica do sistema é lenta.

1.2.1.2 Mesa Instrumentada - Test-bed

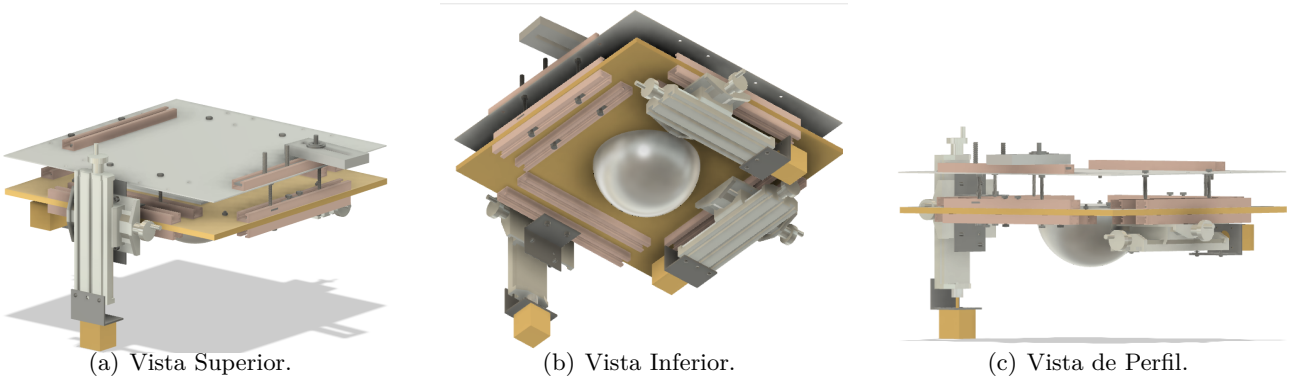
A primeira versão da mesa instrumenta [2] e a sua atualização [3] foram desenvolvidas com um OBC para a mesa com uma IMU mpu9250 e um arduino uno, os aspectos mecânicos podem são exemplificados pelos cads nas Figuras 1.3 e 1.4. Na configuração atual, mudou-se algumas características da mesa como aspectos mecânicos e atualizou-se para um Raspberry Pi o OBC, tais características podem ser exemplificadas no contraste entre as Figuras 1.3 e 1.8.

Na mesa instrumentada encontra-se o OBC, que é a central de processamento embarcada do *Cubesat*. Dentre as arquiteturas de hardware mais usadas atualmente, pode-se destacar as que o processamento é embarcado ou feito pela unidade de processamento central. No caso do simulador, para propiciar as duas características na reformulação da eletrônica [4], um Raspberry Pi 3B (RPI) foi utilizado como a CPU da mesa instrumentada, como mostrado nas Figuras 1.5 e 1.6, sendo responsável pelo sistema de atuação por meio de rodas de reação e o acionamento dos motores de passo para o balanceamento da mesa com as massas móveis e a leitura da IMU.



(a) Vista Superior. (b) Vista Inferior. (c) Vista de Perfil.

Figura 1.3: Vistas da primeira versão da mesa [2] instrumentada onde as estruturas projetadas são as massas móveis.



(a) Vista Superior. (b) Vista Inferior. (c) Vista de Perfil.

Figura 1.4: Vistas da segunda versão da mesa instrumentada [3] desenvolvida com a tampa posta para suportar o *Cubesat* de teste (*Mockup*) em seu topo.

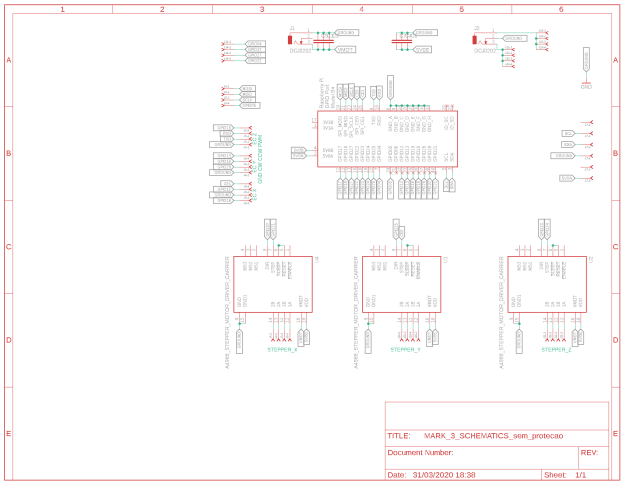


Figura 1.5: Esquemático elétrico do OBC da mesa instrumentada, denominado por MARK III para seguir o padrão estabelecido anteriormente. Fonte: referência [4].

Como relatado na referência [4], o principal motivo para a troca do embarcado foi a integração com o protocolo wifi IEEE 802.11 e o maior poder de processamento fornecido pelo RPI, de forma que pode-se utilizar mecanismos de paralelização e sincronismos como threads e mutex.

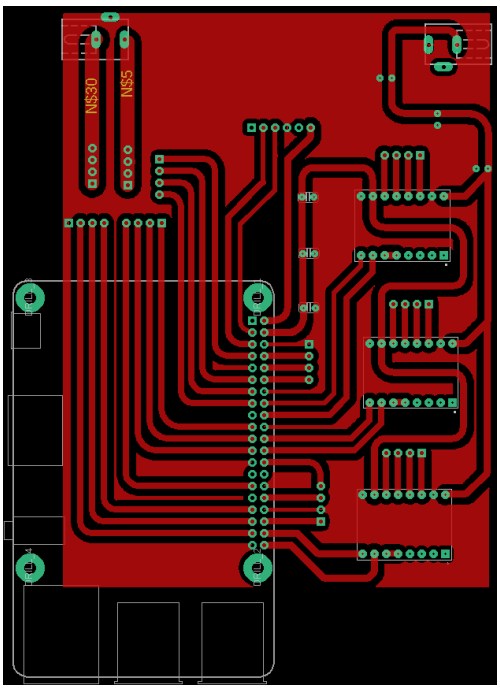
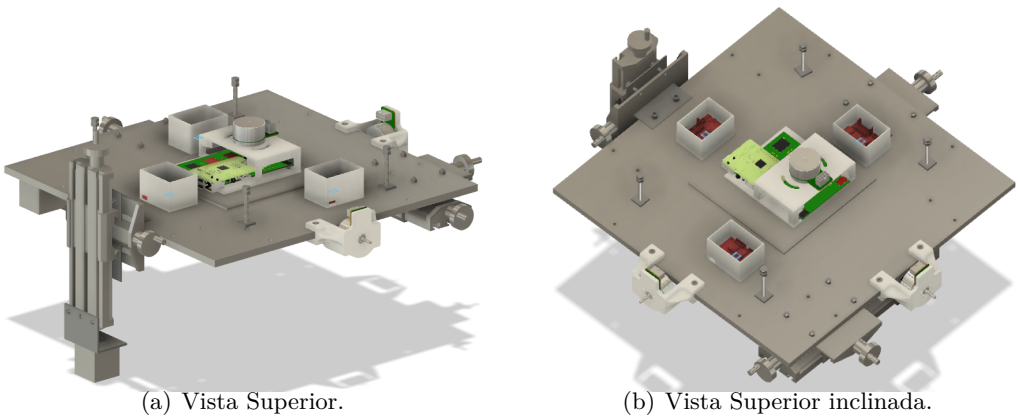


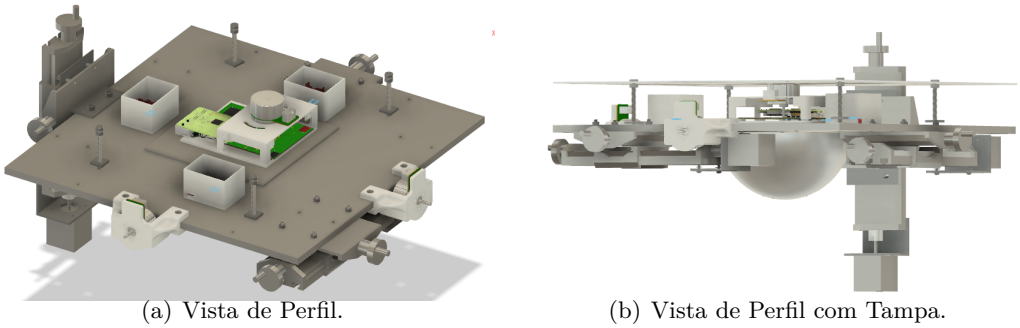
Figura 1.6: PCB desenvolvida para atuação dos motores DC (rodas de reação), dos motores de passo (massas móveis) e acesso ao sensor IMU. Fonte: referência [4].

Além da mudança do OBC, otimizou-se a estrutura mecânica da mesa instrumentada a fim de se diminuir o a massa total e a inércia do sistema, como pode ser visualizado na Figura 1.7. Reduziu-se não só a quantidade de porcas e parafusos, mas também de hastes metálicas utilizadas para fixação das partes que compõem a mesa instrumentada, de forma a massa total da mesa diminuir de 14.550 kg para 10.890 kg.



(a) Vista Superior. (b) Vista Superior inclinada.

Figura 1.7: Vistas da versão mais recente da mesa instrumentada [4] desenvolvida sem a tampa.



(a) Vista de Perfil. (b) Vista de Perfil com Tampa.

Figura 1.8: Vistas da versão mais recente da mesa instrumentada [4].

1.2.1.3 Matrizes de Inércia estimadas via CAD

A primeira mesa com a eletrônica embarcada Mark I, vide referência [3], conforme a Figura 1.4, com massa total de 13.901 Kg distribuida de forma a se ter a matriz de inércia obtida na Equação (1.1). Dado que o tensor de inércia depende do referencial adotado, como definido pela Equação (2.5), o referencial utilizado se dá pelo centro de massa e os eixos X, Y e Z como nos sentidos dos motores DC e da massa móvel Z, vide Figura 1.7. Assim, obteve-se a matriz referente a Equação (1.2), cuja massa total de 10.890 Kg.

$$J = \begin{bmatrix} 0.265 & -0.014 & -0.035 \\ -0.014 & 0.246 & -0.018 \\ -0.035 & -0.018 & 0.427 \end{bmatrix} [kg \cdot m^2] \tag{1.1}$$

$$J = \begin{bmatrix} 0.226 & -0.019 & -0.002 \\ -0.019 & 0.137 & -0.013 \\ -0.002 & -0.013 & 0.311 \end{bmatrix} [kg \cdot m^2] \tag{1.2}$$

1.2.1.4 Unidade central de processamento - ADCV

A unidade central de processamento (estação solo) dentro do laboratório LODESTAR é simbolizada pelo computador mostrado na Figura 1.9. Assim, o computador pode exercer a função apenas do software de determinação de atitude ADCV ou de controle. Vale ressaltar que para sistemas em tempo real, a execução do algoritmo de controle deve ser avaliada a fim de se determinar se essa tarefa pode influenciar nas características de tempo real do ADCV, que será discutida nos Capítulos 3 e 4. O funcionamento do software será melhor detalhado nas sessões subsequentes, em especial na Sessão 1.3.0.2.

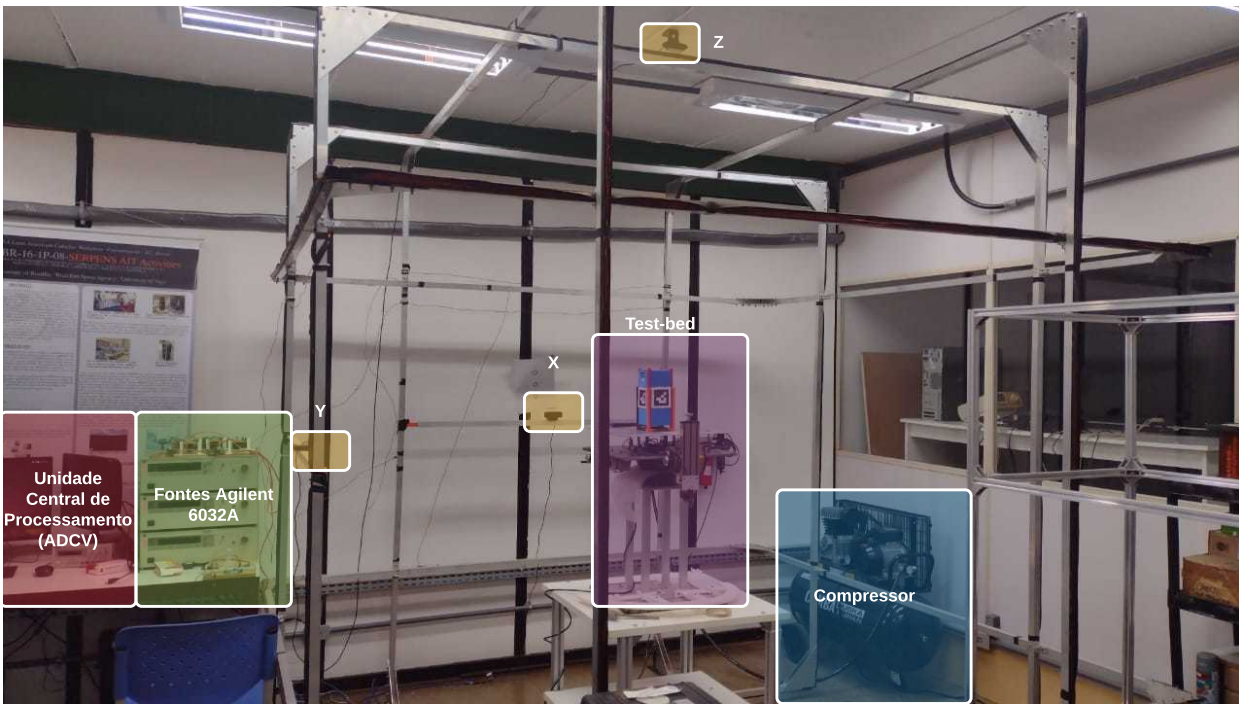


Figura 1.9: Figura representativa das estruturas dentro do laboratório LODESTAR. Em vermelho, no canto esquerdo inferior, tem-se a estação solo representada pelo computador de aplicação que rodará o ADCV. Ao lado, em verde, tem-se as fontes Agillent 6032A as quais são responsáveis por gerar o campo magnético do ponto de órbita por meio de algoritmos de controle e o propagador orbital com feedback pelo magnetômetro HMR2300. Em amarelo tem-se as câmeras Y, X e Z respectivamente. Em roxo a mesa intrumentada (*Test-bed*) sobre o mancal a ar. Por fim, em azul o compressor que promove a sustentação da mesa no mancal a ar.

1.2.2 Arquitetura de Software do Simulador

A arquitetura que define o comportamento do simulador de pequenos satélites foi adaptada a fim de comportar novos módulos e facilitar a implementação e manutenção de todo o sistema.

1.2.2.1 Estrutura do Simulador - Mark I

O sistema do simulador [14, 13, 10, 8] era composto por 2 computadores e dois arduinos Mega (Figura 1.10). O PC1 era responsável pela propragação orbital e o sistema do ADCV (Attitude Determination by Computer Vision), e o PC2 por gerar a ação de controle que seria executada pelo *Mockup*. O dispositivo de comunicação entre subsistemas era o Xbee.

O ciclo de missão da arquitetura pode ser exemplificado de forma macro da seguinte forma:

- No Matlab, o PC1 gera o valor de corrente nas bobinas da Gaiola de HelmHoltz a partir da leitura do magnetômetro HMR-2300 posicionado no centro da gaiola;

- O valor de corrente correspondente é enviado para as fontes que estão conectadas aos enrrolamentos das bobinas, produzindo um campo magnético resultante;
- Paralelamente, no Visual Studio®, há a comunicação com um servidor TCP-IP em Loobback que processa o frame da câmera no eixo Z e envia os dados de atitude estimados para a IDE. Estes valores são enviados pelo Xbee para a mesa (*air bearing table*);
- Após o recebimento dos dados, o Arduino mega coleta os dados da IMU e calcula um valor de erro de atitude, considerando a atitude determinada pelo ADCV como a verdadeira (*ground truth*).
- Em seguida, é enviado por Xbee para o PC2, onde roda o controlador e a ação de controle é gerada, simulando a atuação pela estação solo. A ação é enviada para o Arduino Mega presente na parte superior da mesa, o qual exerce o papel de OBC do *Cubesat*, comunicando com o *Mockup* 2U e atuando os magnetorques.

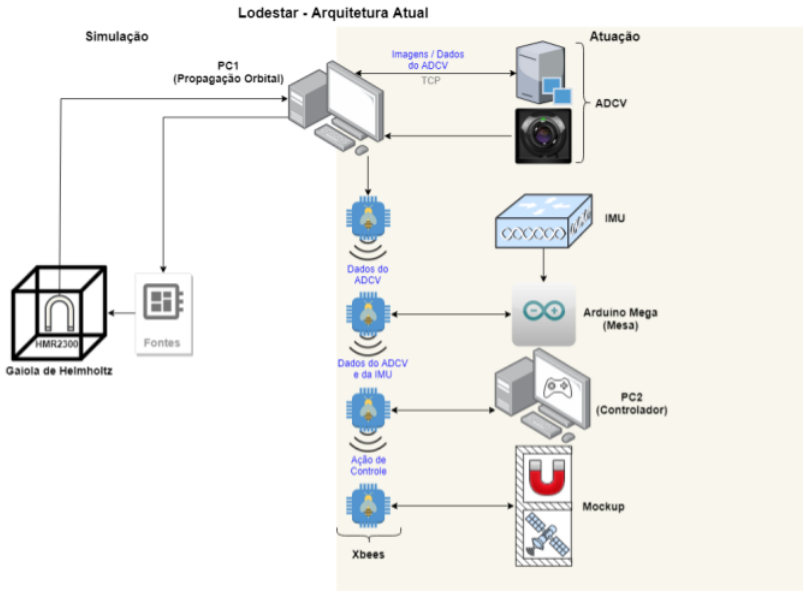


Figura 1.10: Esquemático da antiga arquitetura do Simulador de Pequenos Satélites.

Um ponto a ser destacado é que nesse sistema não havia a implementação do atuador por roda de reação na mesa nem nos satélites (*Mockups*) projetados no LODESTAR. Primeiramente foi implementado como um módulo externo em acrílico e depois incorporado em um *Mockup* 1U apenas para controle em azimute. Dessa forma, tanto na mesa quanto nos *Mockups*, a implementação desse atuador, por exemplo, ou a troca dos módulos Xbee se dava de forma custosa.

1.2.2.2 Estrutura do Simulador - Mark III

A nova estrutura consiste na otimização dos componentes utilizados de forma a explorar recursos dos sistemas operacionais, como Threads e Semáforos, e a integração de módulos por meio

de bibliotecas, ao invés da incorporação direta no código, como era realizado anteriormente, seguindo padrões de engranharia de software. A complexidade do sistema aumentou, apesar da modularidade e escalabilidade do sistema ser possível.

Dessa forma, o novo sistema (Figura 1.11) consiste na separação de módulos: simulação e atuação. Além disso, a sistematização permite o uso de recursos em diferentes tempos, podendo ser extraído a máxima capacidade dos componentes na hora dos testes, ou seja, dividiu-se o teste em três etapas fundamentais: calibração do sistema, setup e missão. Este processo pode ser exemplificado da seguinte forma:

- A etapa de calibração não tem requisitos de sistema em tempo real, uma vez que o ambiente é ajustado para a realização da missão. Assim, a mesa é calibrada com o auxílio dos motores de passo e massa móvel ou manualmente, a IMU é calibrada e, futuramente, o tensor de inércia da mesa com e sem o *Mockup* da missão será estimado, ajustando automaticamente a função de transferência da gaiola. Esta etapa será programada na linguagem Python;
- Na etapa de Setup, os arquivos com os parâmetros dos controladores projetados, assim como as constantes de calibração e da mesa, são enviados para o *Cubesat*. No caso da atuação do controlador ser embarcada, as flags de controle embarcado estarão ativadas e este rodará no Raspberry Pi, caso contrário, será rodado pela estação solo, seguindo o modelo de *Hardware in the loop*, como na referência [10]. Além disso, um ambiente é configurado no Raspberry Pi presente na mesa com todos os arquivos necessários para a missão via SSH (Python);
- Durante a missão, etapa com requisitos de tempo real, o processamento do computador fica exclusivo para o sistema de ADCV (C++) que comunica, via TCP, com o Raspbery Pi 3B (RPI) na mesa. Assim, os dados de atitude gerado pelas câmeras em 3 eixos é comparado com os dados da IMU capturados pelo RPI (C) para a ação do controlador embarcado. No caso em que a ação de controle é gerada pela estação solo, o computador recebe os dados de telemetria do *Mockup* e gera a ação de controle correspondente, levando em consideração os atrasos medidos na fase de calibração, que será enviada para os atuadores conectados ao RPI da mesa.

1.3 Estrutura do Software

O software pode ser dividido em dois módulos, o MCU que é um Raspberry Pi 3b e o ADCV que é o sistema de determinação de atitude por visão computacional, o qual é executado em um desktop, representando a estação solo.

1.3.0.1 MCU

A MCU é a central de processamento do nano satélite, que no caso do simulador, pode ser encontrada na mesa de testes ou no interior do *Cubesat*. Nos projetos que serão mostrados a MCU se encontra na mesa instrumentada, devido a forma como o *Mockup* 2U [9] foi projetado.

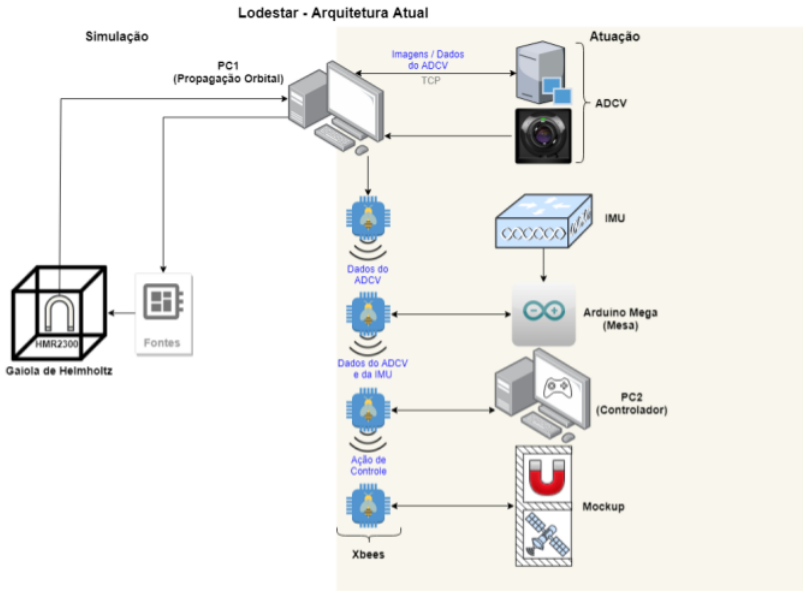


Figura 1.11: Esquemático da nova arquitetura do Simulador de Pequenos Satélites

Após a inicialização dos sensores e conexão TCP com o ADCV, o fluxo do programa executado no Raspberry Pi segue o apresentado na Figura 1.12. A fim de não se ter atrasos na geração da ação de controle, os dados de atitude determinados pelo ADCV são recebidos via TCP e a aquisição dos dados da IMU são executados em uma thread a parte do fluxo principal. Dessa forma, com a paralelização dos processos, o tempo de ciclo do código consegue ser garantido e com o uso de semáforos a sua sincronização é realizada.

O controlador será executado em atraso, ou seja, será computada a ação de controle referente ao dado adquirido no loop anterior e concorrente aos dados atuais. Essa arquitetura será validada no Capítulo 4, a partir de uma análise estatística dos tempos de ciclo do programa para os cenários do computador de projeto, apenas em comunicação loopback, e no computador do laboratório.

1.3.0.2 ADCV

O ADCV é o sistema de determinacação de atitude por visão computacional. A partir do padrão Aruco, as câmeras estimam a matriz de rotação do sistema por meio da Equação (2.4), projetando os pontos da imagem (câmera) para o mundo 3D e consequentemente os ângulos de Rodrigues que são transformados em Euler ou Quaternions.

As etapas de estimação são feitas nos 3 eixos X, Y e Z já que a precisão é maior dos ângulos paralelos ao eixo da câmera [1], ou seja, cada câmera gera um conjunto de ângulos no referencial de Euler [17] φ, ψ e θ em que apenas o ângulo θ é utilizado. As componentes φ, ψ de cada câmera são descartadas devido a grande influência das distorções radias e translacionais intrínsecas aos sensor e que são minimizadas com a calibração do mesmo.

Em paralelo a estimação do Aruco, como mostrado na Figura 1.13, são enviados a atitude estimada no loop anterior via TCP e gravados em um arquivo Json para se ter o logging da

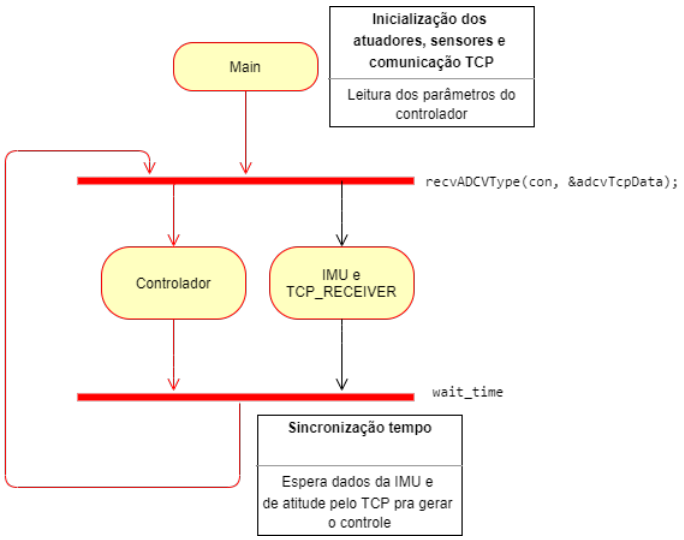


Figura 1.12: Fluxograma do código **main_controller.c** com relação a paralelização por threads e o controlador.

simulação. Nessa versão do ADCV busca-se a questão de paralelismo dos frames, mas não se tinha preocupações com as questões de tempo real e sincronização das threads, solução resolvida no projeto do ADCVRT exposto no Capítulo 3.

Em suma, há o processamento dos frames e a estimação da atitude do *Cubesat* usando o padrão de Aruco nas threads respectivas a cada câmera. Paralelamente, a partir do segundo frame, se tem o envio da componente θ de cada câmera, compondo a tríade φ, ψ e θ de orientação do corpo, por meio do protocolo TCP para o RPI e o armazenamento desses valores em um arquivo no formato JSON para log do experimento, vide Figura 1.13. A sincronização é realizada por evento, ou seja, o maior tempo de processamento entre as câmeras define um loop de execução variado do ADCV.

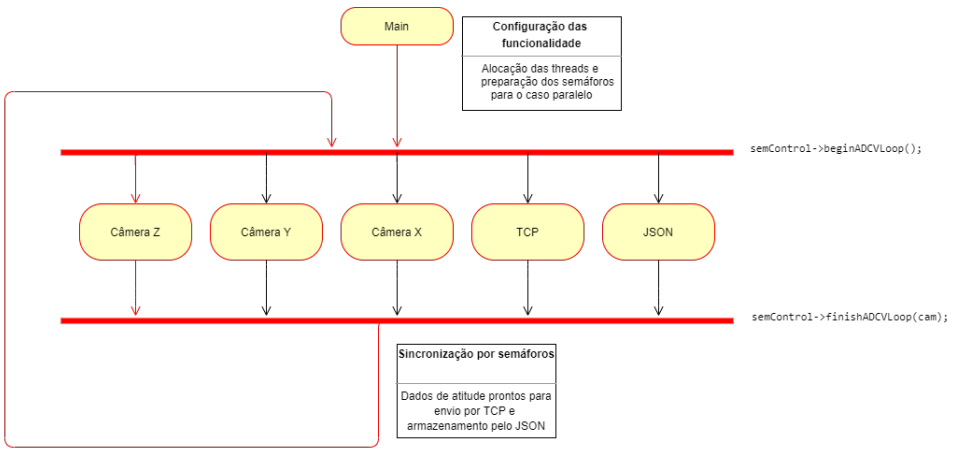


Figura 1.13: Fluxograma do código **main.cpp** com relação a parelilazação por threads e sinconização por semáforos.

A relação de dependência entre os submodulos do ADCV (comunicação, visão computacional, logging e sincronização), que são as classes no paradigma de orientação a objetos, é representada na

Figura 1.14. Assim, foi criado um código exemplo que integra todos esses modulos e submódulos. Vale ressaltar que o módulo TCP usado foi desenvolvido pelo Flávio Amaral e Silva e pode ser encontrado no repositório git [6].

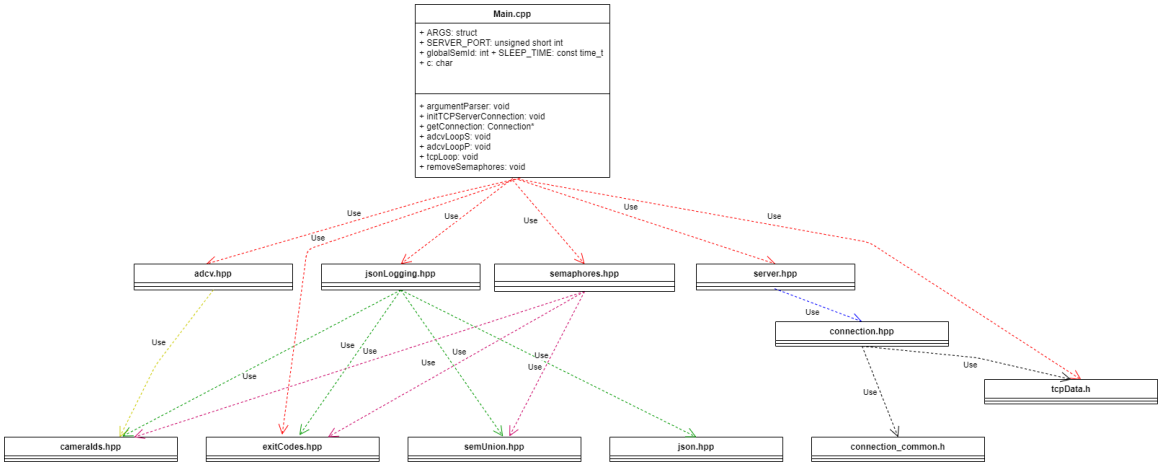


Figura 1.14: Relação de dependência entre bibliotecas e classes da main.cpp

Entretanto, percebe-se que além de diferenças muito grandes entre os ciclos de execução do ADCV, há um atraso total do sistema (*overhead*), que será determinado pelas análises estatísticas do tempo de resposta do ADCV no Capítulo 4. Assim, na tentativa de diminuir esse overhead, as etapas da determinação de atitude, comunicação tcp, logging e UI foram separadas em tarefas e implementadas em threads, seguindo o conceito de escalonador micro kernel, transformando o ADCV em um sistema a tempo real, além da sincronização por tempo máximo de loop e por eventos, objetivando que a média dos tempos de execução se desse próxima de um número inteiro de frames.

1.3.1 Calibração do Simulador

A calibração do sensor IMU e das Câmeras do ADCV foram realizadas em 2019, de forma que não se conseguiu repetir os procedimentos de calibração para documentar as curvas e os dados obtidos. Assim, essa sessão contemplará apenas os procedimentos realizados para a calibração dos sensores.

1.3.1.1 Calibração do ADCV

Conforme o tutorial oficial [18], o framework opensource de visão computacional utilizado no ADCV, necessita-se do padrão de calibração que é um tabuleiro de xadrez contendo 8x9 quadrados, os quais podem variar de dimensão conforme a impressão e o padrão escolhido, como representado na Figura 1.15(a). No processo de calibração da câmera são determinadas as matrizes de coeficientes intrísecos da câmera e as matrizes de distorção radial e translacional, ou seja, as matrizes A, R e t da Equação (2.4).

Dessa forma, pela função *cv::findChessboardCorners()*, as interseções entre os quadrados ou bordas internas são detectadas, formando uma malha com 7x8 pontos que serão rastreados e mapeados na sequência de frames amostrada. Assim, esses pontos são detectados a partir da diferença de 0 e 1 que é formada entre um quadrado preto e branco e os parâmetros estimados são otimizados utilizando métricas de otimização como o algoritmo PnpRansac e o erro de reprojeção dos pontos por meio da Equação (2.3).

Assim, no contexto do laboratório, coloca-se o padrão de calibração o mais próximo das posições onde o padrão Aruco ficaria de forma que o padrão esteja o ortogonal ao eixo da câmera, ou o mais próximo possível. Foram utilizadas trenas para ajudar no alinhamento entre as câmeras e o padrão de calibração. Na Figura 1.15, pode-se ter uma referência da disposição do padrão com relação à câmera Z.

(a) Padrão de Calibração.

(b) Padrão Aruco.

Figura 1.15: Na figura 1.15(a) tem-se a vista da câmera Z do padrão de calibração na mesma distância do do padrão Aruco sobre o *Cubesat* (*Mockup*) 2U. Já, na Figura 1.15(b), tem-se a vista da câmera Z do padrão Aruco sobre o *Cubesat* (*Mockup*) 2U.

1.3.1.2 Calibração IMU

Para a calibração da IMU seguiu-se o tutorial oficial da fabricante Adafruit. De forma resumida, consiste na disposição do sensor de modo a se ter os máximos e mínimos de leitura dos sensores magnetômetro e acelerômetro deixando os eixos do sensor paralelo e ortogonal ao polo norte e a gravidade com respeito ao sistema de coordenadas do sensor (BFF). Assim, automaticamente gera-se o offset e as curvas de calibração pelo firmware disponibilizado.

Dado que a calibração é manual, faz-se importante destacar a repetição do processo a fim de se ter uma média dos valores de offset que podem ser gravados na memória interna da IMU. No caso em questão, foi realizado o procedimento 15 vezes para se ter maior confiabilidade e uma distribuição dos erros associados no processo.

1.4 Definição do problema

Dada a crescente demanda pela validação de algoritmos e a realização de testes com baixo custo, menor que de uma missão, a necessidade de reformulação e padronização do software e hardwares do simulador se tornou evidente. Assim, a arquitetura do simulador foi atualizada de forma a contemplar uma otimização no uso dos subsistemas, inserir sincronismo na transmissão dos dados, além da definição de novos procedimentos para elaboração dos testes garantindo repetibilidade dos ensaios.

1.5 Objetivos do projeto

O objetivo geral deste trabalho é reformular o projeto da mesa de forma a inserir requisitos temporais, como a estimação de um tempo de loop do ADCV, atrasos do sistema, sincronização dos dados, estimação dos erros das variáveis medidas e estimadas pelo ADCV. Especificamente, a nova plataforma será implementada como um sistema em tempo real, e, neste contexto, o sistema de determinação de atitude por visão computacional será validado. Assim, será realizado o projeto do sistema de determinação de atitude por visão computacional em tempo real (ADCVRT), os testes realizados para a validação do sistema a partir da estimação do WCRT e o teste para determinação da acurácia da atitude determinada pelo ADCVRT em contraposição a referência e ao sensor inercial (IMU).

1.6 Apresentação do manuscrito

Essa dissertação é divida em 6 capítulos principais, onde no Capítulo 2 é exposto os conceitos trabalhados na elaboraçãos dos projetos e dos módulos do sistema que compõem o simulador de pequenos satélites; no Capítulo 3 mostra-se o projeto da arquitetura final do ADCV para um sistema em tempo real; no Capítulo 4 são explicitados os resultados dos testes realizados para a validação do ADCV-RT no computador de projeto e na unidade central de processamento, tanto para a comunicação em loopback quanto para a comunicação TCP entre o ADCVRT e o Rapberry Pi 3b (RPI);no Capítulo 5 são apresentados a análise dos erros e precisão do ADCVRT quanto a determinação de atitude. Por fim, as conclusões e perspectivas futuras são apresentadas no Capítulo 6.

Capítulo 2

Fundamentos

Esse Capítulo apresenta a fundamentação teórica necessária para a realização do projeto, assim como os conceitos envolvidos para a elaboração do mesmo, vide referências [17, 19, 20].

2.1 Sistemas de referência

Dentre os diversos tipos de representação da orientação do corpo em um sistema de coordenadas, será apresentado apenas o ORB e o BFF devido ao fato de serem os sistemas de referência utilizados no projeto do simulador. As principais referências bibliográficas para essa sessão são [17, 21, 9, 5].

2.1.1 BFF (*Body-fixed frame*)

É o referencial do corpo. Conforme a referência [17] pode ser especificado como qualquer ponto com três eixos cartesianos na estrutura do corpo analisado. O centro desse sistema de coordenadas pode ser fixado em qualquer ponto no interior do satélite e seus eixos sejam ortogonais, unitários e obedeçam a regra da mão direita. Para os casos investigados nesse trabalho, a origem do sistema será definida no centro geométrico do nano satélite, como mostrado na Figura 2.1 no caso de um *Cubesat* 3U, com os eixos paralelos aos eixos definidos pelo sensor de unidade inercial (IMU). Assim, esse referêncial é não inercial, de forma que as componentes dos vetores de aceleração e velocidade angulares medidas por sensores do tipo IMU, por exemplo, são afetadas pela movimentação desse sistema de referência.

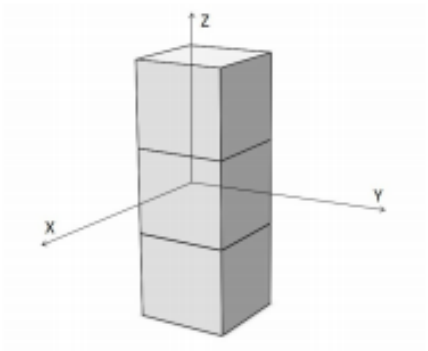


Figura 2.1: Figura representativa do referêncial no corpo em órbita e amplificado no mockup. Fonte: referência [5].

2.1.2 ORB (*Orbit reference frame*)

É o sistema de referência de acordo com a órbita terrestre. Conforme as referências [20] com a origem centrada no centro de massa do corpo e o eixo Z apontando no sentido radial para o centro de massa da Terra, vide Figura 2.2. O eixo Y é definido como o vetor antiparalelo ao vetor normal ao plano orbital. Assim, o eixo X é definido pela tríade formada pela regra da mão direita.

Esses sistemas de coordenadas serão representados posteriormente, uma vez que o BFF é o do *Cubesat* (IMU) e o ORB é o da gaiola de Helmholtz (ADCV). Vale ressaltar que o eixo Y foi escolhido no sentido antiparalelo a normal devido a ser o sentido natural da câmera Z utilizada no caso investigado nesse trabalho.

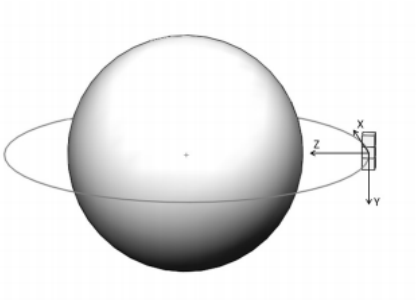


Figura 2.2: Figura representativa do referêncial em órbita do mockup. Fonte: referência [5].

2.2 Matriz de rotação

A determinação de atitude vem da necessidade de se representar os vetores do seu sistema de coordenadas original para outro escolhido. Tal processo se dá pela matriz de rotação (R) referente a esses sistemas, ou seja, a multiplicação do vetor u_1 em um sistema de coordenadas pela matriz de rotação gera o vetor u_2 correspondente no sistema de coordenadas desejado, de acordo com a

Equação (2.2) para sistemas de coordenadas 3d. A operação entre os vetores é definida como o produto interno pela Equação (2.1), caracterizando a matriz de rotação como a matriz de cossenos diretores.

$$\cos\alpha = \langle u_1, u_2 \rangle = u_1^T u_2. \tag{2.1}$$

$$\begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} = \begin{bmatrix} \langle x_1, x_2 \rangle & \langle x_1, y_2 \rangle & \langle x_1, z_2 \rangle \\ \langle y_1, x_2 \rangle & \langle y_1, y_2 \rangle & \langle y_1, z_2 \rangle \\ \langle z_1, x_2 \rangle & \langle z_1, y_2 \rangle & \langle z_1, z_1 \rangle \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix}. \tag{2.2}$$

2.2.1 Matriz de rotação do 2D para o 3D

Quando se trata da determinação de atitude por visão computacional, em que os pontos de uma imagem (2d) são projetados no mundo físico (3d), tem-se a Equação de transformação desses sistemas dada pela Equação (2.3). Assim, a equação (2.4) é a forma simplificada da Equação (2.3), onde a matriz A corresponde a matriz dos parâmetros da câmera (obtida pela calibração da mesma); a matriz *[R/t]* é a matriz de rotação concatenada com a matriz de translação do sistema, as quais serão estimadas por meio do padrão Aruco; e as matrizes P são os pontos em 2D e 3D respectivamente.

$$\begin{bmatrix} x_{2d} \\ y_{2d} \\ 1 \end{bmatrix} = \frac{1}{s} \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} x_{3d} \\ y_{3d} \\ z_{3d} \\ 1 \end{bmatrix}. \tag{2.3}$$

Vale ressaltar que as variáveis **f, c, r** e t correspondem a distância focal, aos centros geométricos da lente da câmera e as componentes dos vetores de rotação e translação do sistema.

$$P_{2d} = A \times [R|t] \times P_{3d}. \tag{2.4}$$

Assim, no que condiz ao sistema do ADCV (*Attitude Determination by Computer Vision*), o qual é responsável pela referência do simulador, os pontos encontrados na imagem são transformados em pontos no mundo físico a partir da Equação (2.3), sendo previamente calibrada a câmera ao se colocar o padrão de calibração sobre o plano que o Aruco ficará. Um exemplo de saída desse sistema pode ser visto na Figura 4.1

2.2.2 Atitude de um corpo

A atitude de um corpo pode ser parametrizada como uma matriz de rotação entre bases de espaços vetoriais distintos, ou referenciais, de forma a rotacionar um vetor de um sistema de referência para outro, vide referência [20]. A atitude pode ser representada em diferentes sistemas

ou notações, como Euler, Rodrigues e Quaternions, por exemplo. A mudança da atitude de um referêncial para outro, mesmo que de espaços vetoriais distintos, se dá pela matriz de rotação ou atitude entre esses sistemas.

2.3 Equações dinâmicas da mesa

Conforme apresentado em [19], o momento de inércia pode ser definido pela Equação (2.5), onde a variável r representa a distância (braço de alavanca) do elemento infinitesimal dm até o eixo de referência analisado. Por comparação ao caso linear, pela segunda Lei de Newton, pode-se dizer que o momento de inércia é uma medida de resistência de um corpo a uma aceleração ângular.

Assim, a matriz de inércia (J) de um corpo rígido por ser definida como uma matriz 3x3, descrita na Equação (2.6), onde a diagonal principal é a relação de inércia referente a cada eixo X, Y e Z e os demais elementos a interferência dessas componentes sobre as outras componentes, ou seja, o momento de inércia cruzado.

$$J = \int_m r^2 dm. \tag{2.5}$$

$$J = \begin{bmatrix} J_{XX} & J_{XY} & J_{XZ} \\ J_{YX} & J_{YY} & J_{YZ} \\ J_{ZX} & J_{ZY} & J_{ZZ} \end{bmatrix}. \tag{2.6}$$

Como mostrado na Sessão 1.2.1.3, as componentes cruzadas do tensor de inércia são ligeiramente menores que as componentes da diagonal principal (momentos principais de inércia) devido a geometria e o projeto da mesa instrumentada e do *Cubesat*, podendo-se desprezar as componentes cruzadas. Dessa forma, tomando-se o referêncial dos eixos princiais de inércia, obtem-se a matriz exposta na Equação (2.7). As componentes J_{XX}, J_{YY} e J_{ZZ} podem ser definidas conforme a Equação 2.5, onde x, y e z correspondem às distâncias aos eixos X, Y e Z, respectivamente.

$$\begin{array}{l} J_{XX} = \int_m (y^2 + z^2) dm \\ J_{YY} = \int_m (x^2 + z^2) dm. \\ J_{ZZ} = \int_m (x^2 + y^2) dm \end{array} \tag{2.7}$$

Dessa forma, como exposto nas referências [20, 19, 2, 3] o equacionamento dinâmico de um pêndulo invertido, que é a modelagem da dinâmica da mesa instrumentada (*test-bed*) sobre o mancal a ar, pode ser descrito na Equação (2.8).

$$\dot{\omega} = \begin{bmatrix} \frac{-mg}{J_{XX}}(r_y c\phi c\theta - r_z s\phi c\theta) \\ \frac{mg}{J_{YY}}(r_x c\phi c\theta + r_z s\theta) \\ \frac{-mg}{J_{ZZ}}(r_x s\phi c\theta + r_y s\theta) \end{bmatrix}. \tag{2.8}$$

2.4 Protocolos de comunicação

O simulador de pequenos satélites é composto por três principais módulos: gaiola de Helmholtz, a mesa instrumentada e o computador usado como unidade principal de processamento que simboliza a estação solo. Tais módulos trocam informações entre si seguindo o conceito de *Hardware in the loop - HIL* [9]. Assim, os principais protocolos utilizados são o TCP, protocolo de redes para comunicação entre a unidade de processamento (estação solo) e a mesa intrumentada, e o I2C entre o OBC (*Onboard Computer*), que nesse caso é um Raspberry Pi modelo 3b, e a IMU.

2.4.1 Protocolos de rede

Dentre os protocolos de rede utilizados, pode-se destacar o TCP (*Transmission Control Protocol*) e o SSH (*Secure Shell*). O primeiro é utilizado para a comunicação entre o ADCV e a mesa instrumentada durante o experimento, já que se necessita de confiabilidade na comunicação e bom desempenho, especificamente, a mensagem necessita chegar o mais rápido possível. Já o SSH é utilizado para configuração do ambiente, juntamente com o protocolo SFTP (*Secure File Transfer Protocol*) para upload dos arquivos na fase de preparação dos testes (*setup*).

2.4.1.1 Protocolo TCP

A especificação ISO divide a rede de computadores em 7 (sete) camadas: transporte, sessão, aplicação, apresentação, rede, enlace e físico (Figura 2.3). O protocolo TCP é um protocolo da camada de transporte que, através da camada de rede, age sobre o protocolo IEEE 802.11 (wifi) da camada de enlace. Esse protocolo é orientado à conexão, ou seja, ele estabelece a conexão e só a desfaz quando não recebe respostas da outra ponta de comunicação após um intervalo de tempo ou a comunicação é encerrada por uma das duas pontas.

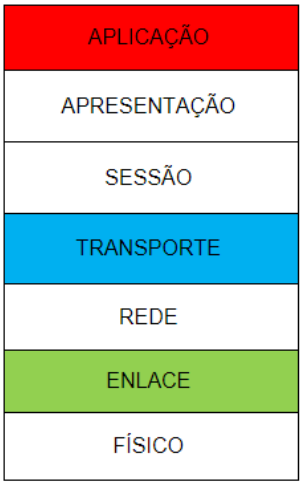


Figura 2.3: Figura representativa da abordagem topdown do modelo OSI (Open Systems Interconnection).

O protocolo TCP/IP funciona com o paradigma cliente-servidor. Ao conectar-se, os dois processos enviam segmentos preliminares um ao outro para estabelecer os parâmetros da transferência de dados antes de realizar as trocas de informação. Importante destacar que a comunicação via TCP é full-duplex e ponto a ponto, ou seja, informações podem ser enviadas de ambos os lados sem que abra uma nova conexão e a comunicação é somente entre as entidades envolvidas. Para tanto, durante a transmissão, um pacote enviado necessita da sua confirmação acerca da chegada e da integridade dos dados, sendo reenviada quando não há essa confirmação após determinado tempo, como ilustrado na Figura 2.4. Para fechar a conexão, um dos lados, servidor ou cliente, informa a outra ponta essa necessidade e espera a confirmação, de modo a garantir que a conexão será fechada em ambas as pontas.

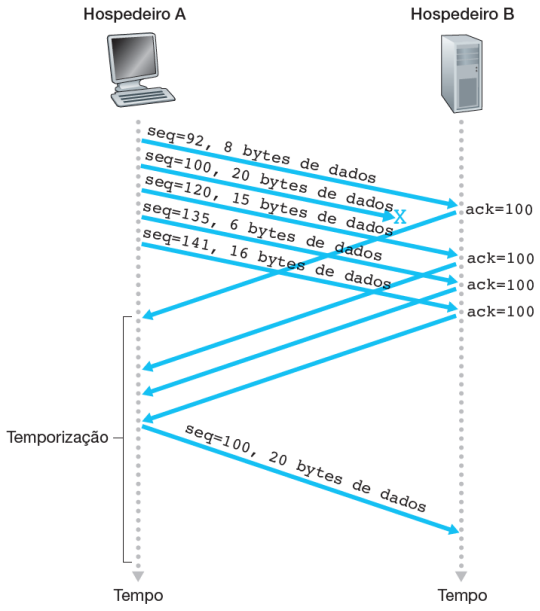


Figura 2.4: Figura 3.37 página 183 do livro Redes de Computadores e a Internet [6], na qual é apresentada a transmissão de dados pelo protocolo TCP entre cliente (A) e servidor (B).

Dessa forma, é necessário distinguir o protocolo TCP da camada de transporte com o TCP-sockets da camada de aplicação. Esse protocolo se baseia na solução da camada de transporte, entretanto, a verificação de mensagens é feita apenas na camada de transporte.

2.4.1.2 Protocolo SSH

O protocolo SSH é um protocolo da camada de aplicação que usa o protocolo TCP para trocar informações entre cliente e servidor, mas com uma camada de criptografia durante a etapa de conexão, tornando os dados mais seguros e confiáveis contra ataques de rede. Assim, um par de chaves criptografas é mantido no servidor e no cliente, de modo que, no início da conexão, há a requisição dessa chave por parte do cliente e a sua consecutiva validação. Caso esteja correta, abre-se esse canal seguro de comunicação ponto a ponto e o cliente conecta-se ao servidor a partir da chave. Em seguida, a comunicação segue similar ao protocolo TCP, mas com a criptografia das

mensagens por meio de algoritmos hash.

O SFTP é um protocolo de transferência de arquivos ou diretórios (pastas) tendo por base o protocolo SSH, ou seja, a questão da correspondência entre as chaves públicas (do cliente) e privadas (do servidor) e a criptografia das mensagens, conforme a referência [22].

2.5 Sistema a tempo real

Sistemas a tempo real (STR) [7] é um sistema computacional capaz de processar dados e gerar uma resposta logicamente correta respeitando um requisito temporal pré-determinado para cada situação e determinado pelo tempo do mundo físico. Para entender o sistema e assegurar-se que os requisitos serão cumpridos, usam-se modelos que descrevem como a execução do software do sistema deve proceder, mesmo nos casos de não cumprimento desses requisitos.

Um conceito importante de se destacar é o de tarefa (*task*), que comumente na computação é empregado como a execução de um programa independente, mas em RTOS são considerados como unidades mínimas para cumprir determinado objetivo, ou seja, podem ser funções na linguagem de programação adotada. Faz-se importante ressaltar que linguagens interpretadas como o python não são ideais para tempo real uma vez que as etapas desde o início da execução do programa até a execução de fato das instruções em assembly no processador e a virtualização podem gerar atrasos e apresentar imprevisibilidades que são significativas nesse contexto. Por isso serão utilizadas as linguagens C++ para a unidade de processamento central (estação solo) e C para o RPI.

Ainda, os sistemas em tempo real podem ser classificados quanto os aspectos abaixo:

- **Criticalidade**: Uma tarefa é considerada crítica quando o não atendimento de seus requisitos temporais pode resultar no comprometimento do sistema. O requisito temporal mais usado para determinar a criticalidade de um STR é o Deadline, que é o instante máximo desejado para concluir uma tarefa. Existe o Deadline relativo e absoluto. O primeiro é dado em relação ao início da tarefa e o segundo em relação ao tempo UTC.
- **Periodicidade**: Uma tarefa que é desempenhada periodicamente por um STR deve ser executada dentro de um ciclo. A duração desse ciclo é o período da tarefa. Caso a tarefa não seja periódica, mas exista um intervalo mínimo entre o instante de duas chegadas, é classificada como esporádica. Por fim, se não houver conhecimento temporal de uma tarefa, esta é considerada aperiódica.
- **Previsibilidade**: A previsibilidade antecipa ao desenvolvedor se um STR cumprirá seus prazos. Em geral, usa-se uma previsibilidade determinista, a qual garante se os deadlines serão respeitados, mas também pode ser usada uma abordagem probabilística quando se tem tarefas esporádicas ou aperiódicas.

Outro conceito importante para sistemas em tempo real é o tempo de execução da tarefa, que é o tempo de processador necessário para completar a tarefa, sem considerar interferências.

Em geral, o tempo de execução do pior caso é mais utilizado. Além disso, existe ainda o tempo de resposta que é o intervalo entre a chegada da tarefa e a sua conclusão, incluindo eventuais interferências.

2.5.1 Escalonamento

Em sistemas de tempo real sempre há diversas tarefas concorrendo para serem executadas no processador. O escalonamento é a forma de se organizar a execução dessas tarefas, ou seja, a ordem que as tarefas serão executadas e como os recursos do sistema serão alocados para as tarefas. O escalonador é o módulo do sistema operacional responsável por essa escolha.

O escalonamento pode ser estático, que utiliza informações disponíveis previamente à execução das tarefas, como sua periodicidade, previsibilidade, tempo de execução da tarefa e o seu deadline. Entretanto, o escalonamento pode ser dinâmico, cujas informações estão disponíveis apenas durante a execução do programa, podendo-se organizá-las a depender do algoritmo de escalonamento implementado no escalonador do sistema. Dessa forma, este pode ser orientado por eventos, onde o sistema reage a eventos externos gerando interrupções e disparando tarefas; ou por tempo, onde é controlado apenas pela interrupção de relógio a cada T milissegundos do mundo real.

2.5.2 Relação entre tarefas

Tarefas independentes são aquelas cuja a execução não está relacionada com a outra tarefa analisada em questão, de forma que não há precedência entre as duas. Entretanto, é comum que uma tarefa só possa ocorrer após outra e nesse caso há relação de precedência entre elas. Um conjunto de tarefas interligadas por relações de precedência é denominado atividade.

Pode-se afirmar que há uma relação de disputa ou concorrência por determinado recurso, a saber, o processador ou acesso a memória para a escrita, quando mais de uma tarefa consome aquele recurso que está em uso. Assim, uma tarefa que está na fila de tarefas de um escalonador, pronta pra ser processada, ou em processamento, e necessita de um recurso que está sendo consumido por outra há uma condição de exclusão mútua, como na Figura 2.5. Para tanto, o recurso a ser disputado é chamado de seção crítica e precisa ser protegido por mecanismos de sincronização como MUTEX, semáforos e monitores, que bloqueiam tarefas que tentam acessar seções críticas simultaneamente.

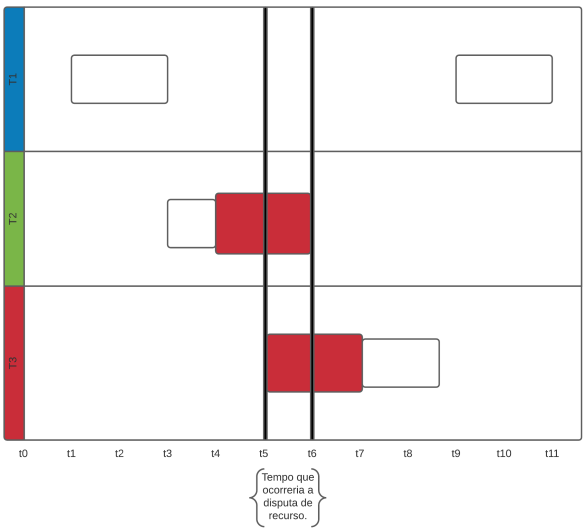


Figura 2.5: Exemplo de concorrência por recurso. Os quadrados brancos representam o tempo de execução daquela tarefa e o quadrado em vermelho o tempo de utilização de determinado recurso. Assim, como apontado, entre os tempo t5 e t6 as tarefas T2 e T3 utilizam o mesmo recurso. A utilização dos mecanismos supracitados, como o MUTEX.

2.5.3 Escalonadores

Os principais tipos de escalonadores são: executivo cíclico, laço principal com interrupções e sistema operacional multitarefa na forma de um microkernel.

- **Executivo Cíclico**: Consiste em um grande laço (ciclo maior) que sempre é repetido periodicamente. O período de repetição desse laço é controlado por um temporizador em hardware, sendo necessário garantir que todas as tarefas executem dentro do laço. Como o período das tarefas varia, é possível usar um ciclo menor para que todos os períodos sejam respeitados. Tipicamente, escolhe-se o MMC dos períodos como o ciclo maior e o MDC é conveniente para a escolha do ciclo menor. Operações de entrada e saída de dados, ou seja, que interagem com o mundo real, devem ser assíncronas, pois a tarefa não fica bloqueada nesse tipo de solução.

- **O Laço Principal com Interrupções**: Esse escalonador acrescenta à metodologia do executivo cíclico os tratadores de interrupções. A interrupção é um recurso que permite que um controlador de periférico, um programa ou o processador possa interromper uma tarefa em execução para tratar de um evento mais importante. Um tratador de interrupção é uma rotina usada pelo processador para tratar das interrupções e salvar os registradores das tarefas interrompidas para continuá-las depois. É possível, ainda, desabilitar interrupções para proteger variáveis globais.

- **Microkernel**: É uma versão básica do kernel, que por sua vez é a parte principal do sistema operacional, o qual controla diretamente os recursos de hardware. Esse escalonador

se utiliza da técnica denominada multiprogramação, onde há a abstração de tarefas em threads, dividindo o tempo do processador e criando, assim, a ilusão que as threads executam simultaneamente, no caso de um único núcleo de processamento. Além disso, as interrupções também são implementadas como threads, que solicitam serviços ao microkernel através de chamadas de sistemas. O microkernel é um sistema reativo, sendo ativado por interrupções de hardware ou software.

Threads podem ser criadas e destruídas dinamicamente por chamadas de sistemas. Após ser criada, um thread espera para usar o processador na fila de aptos do escalonador, onde aguarda o momento que será executada. Uma vez em execução, a thread pode ser bloqueada por requerer algum serviço (concorrência) ou interromper a execução da thread que está usando tal recurso, a depender das deadlines e prioridades entre as tarefas.

No caso em que a tarefa é bloqueada, ela retorna à fila de aptos, podendo "furar"a fila caso haja algum algoritmo de envelhecimento, ou seja, sobe-se a prioridade das tarefas a medida que elas passam mais tempo na fila de aptos. Quando uma thread é bloqueada, faz-se necessário o chaveamento de contexto, que é o salvamento dos registradores, de modo que ela possa continuar de onde parou quando voltar.

2.5.4 Algoritmos de Escalonamento

Para escolher a ordem que as tarefas ou threads executam, o escalonador faz o uso de algum algoritmo escolhido no projeto do sistema com base na criticalidade das tarefas, periodicidade e tempo de execução. Assim, dentre os algoritmos de escalonamento, pode-se destacar:

- **Ordem de Chegada (FIFO)**: A tarefa, ao ficar apta, entra no final da fila e a primeira tarefa dessa fila será a próxima a executar.
- **SJF**: Executa primeiro a tarefa que necessita de menos tempo de processador até ficar bloqueada. Sua implementação é muito difícil e acaba se tornando impraticável devido ao custo computacional de se comparar todos os elementos da fila. Em filas muito extensas pode-se perder o deadline com mais facilidade.
- **Fatias de Tempo (Round-Robin)**: Usa uma metodologia semelhante ao FIFO, só que estabelece uma fatia de tempo máxima para a tarefa executar. Se não liberar o processador antes, o escalonador salva o contexto e a tarefa é posta no fim da fila de aptos.

É possível estabelecer prioridades para as tarefas, de modo que as mais altas executam antes. Diversos fatores podem influenciar na atribuição de prioridades, deadline, período, folga e até mesmo o tempo que a tarefa passa na fila de aptos. As prioridades podem ser fixas ou variáveis, sendo possível aplicar testes de escalonabilidade para saber se os deadlines serão cumpridos durante a fase de projeto do sistema.

2.5.5 WCET

O tempo de execução da tarefa no pior caso (worst case execution time-WCET) é uma informação extremamente relevante para um projeto de aplicação em tempo real. Nele é suposta a ocorrência simultânea de todas as interrupções e atrasos que o hardware e software do sistema podem agregar ao tempo de execução da tarefa. A ocorrência do WCET, no entanto, é extremamente rara, fazendo-se necessário o uso de técnicas para sua estimação, destacando-se a estática e a de medições.

2.5.5.1 Estimação Estática

As técnicas de estimação estática requerem um conhecimento profundo da arquitetura e hardware do sistema onde a aplicação será implementada. Examinam-se fluxo de controle da tarefa e o contexto de execução, onde o primeiro é o caminho de linhas de código que determinadas entradas geram na tarefa, focando em condições e saltos. Já o segundo é um histórico de execução da tarefa que reúne informações sobre o comportamento do processador, tais como: se a posição de memória acessada está na cache, se haverá atraso no pipeline, ou o previsor de saltos acertará, sempre fazendo suposições pessimista em casos de dúvida.

Estimações estáticas são excelentes quando a aplicação é restrita a alguma arquitetura ou hardware, fazendo-se uma estimação de WCET mais concreta. De encontro a isso, a técnica não é recomendável para aplicações portáveis por ser extremamente custosa para ser realizada tornando a estimativa presa ao hardware em questão, além de exigir uma análise profunda por parte do desenvolvedor, de modo que as informações necessárias não são normalmente disponibilizadas pela fabricante da máquina. Vale ressaltar que o tipo de processadores para esse tipo de análise são os de 8 ou 16 bits clássicos, como Texas TMS320F ou o ARM Cortex-M3. Como o RPI possui um processador do tipo Cortex-A53 (ARMv8) 64-bits, não será possível realizar esse tipo de abordagem para a estimação do WCET.

2.5.5.2 Estimação por Medições

Já as técnicas de estimação do WCET por medições realizam análises probabilísticas aplicando a teoria dos valores extremos(TVE) a dados coletados em diversos testes realizados na tarefa, vide o livro do Rômulo Oliveira [7]. Existem duas técnicas de obtenção do WCET probabilístico (pWCET), o primeiro é o máximo de blocos (BM), mostrada na Figura 2.6, onde se particiona em blocos do mesmo tamanho o conjunto de dados amostrados selecionando o máximo de cada bloco, utilizando-se a função generalizada de valores extremos (GVE), conforme a Equação 2.9, que unifica as três distribuições Weibull, Gumbel e Fréchet.

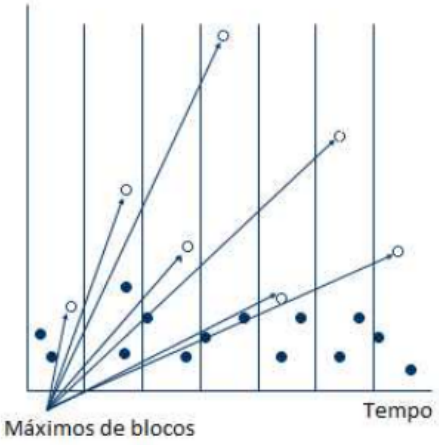


Figura 2.6: No BM, seleciona-se os máximos locais de acordo com as fatias fixas de tempo pré definidas. Após essa seleção aplica-se a função GVE para estimar a probabilidade de ocorrência desses dados. Fonte: referência [7].

$$\begin{aligned} F(X) &= \exp\left(-1 * [1 + \xi(x-\mu)/\sigma]^{1/\xi}\right), \xi \neq 0; \\ F(X) &= \exp\left(-1 * \exp[-1 * (x-\mu)/\sigma]\right), \xi = 0. \end{aligned} \tag{2.9}$$

O segundo é o pico acima do limiar (POT), onde apenas as medições acima de um limiar TH são consideradas e é usada a distribuição generalizada de Pareto (GP), conforme a Equação 2.10, para a estimação da ocorrência desses dados de amostragem, e pode ser visualizada na Figura 2.7, onde a variável u corresponde ao TH. Por exemplo, pode-se selecionar apenas os dados acima de dois desvios padrões da média, de forma a excluir 95% dos dados amostrados e se concentrar nos 2.5% dos dados, seguindo o TVE. Dessa forma, faz-se necessária a repetição do experimento de forma a se ter que os dados selecionados sejam representativos dentro do seu espaço amostral, evitando-se o cenário presente na Figura 2.8.

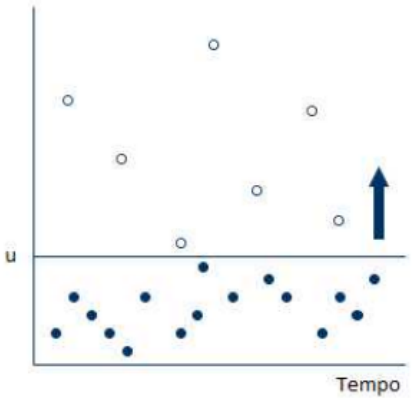


Figura 2.7: No POT, seleciona-se todos os pontos acima de um limiar $u = TH$ e depois aplica-se a função GP para . Figura retirada da referência [7].

$$F(X) = 1 - [1 + \xi((x-\mu)/\sigma)]^{1/\xi}, \xi \neq 0;$$
$$F(X) = 1 - \exp[-1 * ((x-\mu)/\sigma)], \xi = 0. \tag{2.10}$$

Onde o threshold (μ), a escala (σ) e forma (ξ) são apresentadas nas Equações (2.9) e (2.10). No que concerne as condições em que se aplicam a GVE e a GP, tem-se que:

- **GVE**:
  - Weibull: Com $\xi < 0$, com cauda curta delimitada;
  - Gumbel: Com $\xi = 0$, com cauda ilimitada e decrescimento exponencial ($\xi \to 0$);
  - Fréchet: Com $\xi > 0$, com cauda pesada ilimitada.
- **GP**:
  - Com $\xi < 0$ modela caudas leves, limitadas por um valor máximo, semelhante à distribuição Beta;
  - Com $\xi = 0$ modela caudas ilimitadas, cuja densidade decresce exponencialmente, como a distribuição Exponencial;
  - Com $\xi > 0$ modela caudas pesadas assintóticas, cuja densidade decresce polinomialmente no caso de parâmetro ξ elevado, semelhante à distribuição de Pareto.

Em ambos os casos, os valores são ajustados através da PDF e CDF obtidas pelas funções, sendo possível obter um pWCET que tenha baixa probabilidade na curva 1-CDF. As técnicas de obtenção por medições são mais utilizadas na indústria, uma vez que não é necessário um entendimento aprofundado da máquina usada, apenas testes exaustivos e em diversas condições que estão previstas em projeto.

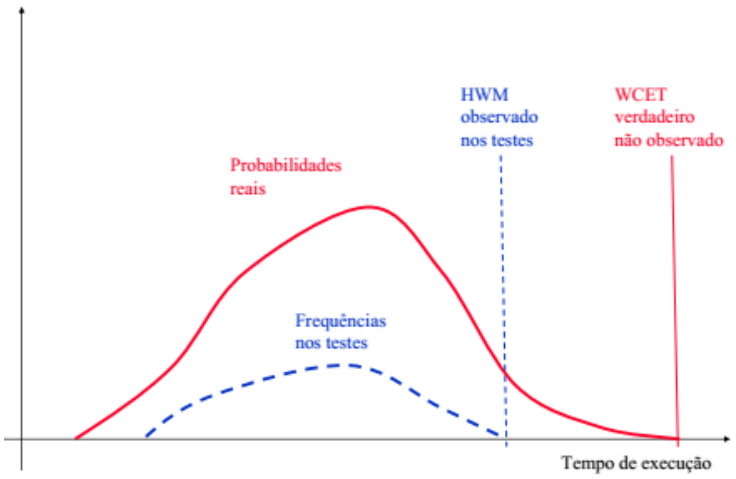


Figura 2.8: A definição do HWM (High Water Mark) se dá pelo máximo local encontrado no seu conjunto de testes, conforme representado na figura. Já o WCET é o pior caso que pode ou não ser coberto dentro da estimativa, a depender do conjunto de dados, ou seja, o máximo global do sistema. Figura retirada da referência [7].

2.5.6 WCRT

O tempo de resposta no pior caso (WCRT) engloba o WCET de uma tarefa e quaisquer bloqueios, interrupções e acréscimos de tempo entre o momento que a tarefa entra na fila de aptos e termina de ser executada. O WCRT é extremamente raro de ocorrer, uma vez que características de hardware, outras tarefas concorrentes e qualquer elemento não controlável pelo desenvolvedor pode influenciá-lo. Por esse motivo que a estimativa usando medições e testes com características probabilísticas é a mais utilizada na prática.

O teste mais usado para se obter uma estimativa do pWCRT é o teste de resposta do sistema a fim de se obter a média, mediana, variância, desvio padrão, mínimo e máximo observados com os dados colhidos. É também feito um histograma com os valores obtidos e adapta-se uma curva probabilística (geralmente uma gaussiana) que se encaixe bem para se obter a probabilidade do máximo observado (high water mark-HWM). Quanto menor for essa estatística, mais próximo do pWCRT será o HWM e, por fim, adiciona-se uma margem de segurança, relacionada com a variância observada, ao HWM para a estimação do pWCRT. No caso será considerada essa margem de segurança como 20% acima do HWM.

Capítulo 3

Projeto do ADCV-RT

O ADCVRT foi baseado no ADCV apresentado na Sessão 1.3.0.2, de forma que as funcionalidades do mesmo fossem implementadas no modelo de tarefas utilizando-se escalonador microkernel. Primeiramente, buscou-se frameworks convencionais de sistemas em tempo real como o FreeRTOS e o ROS2 para a implementação da solução, já que eles abstraem e facilitam a sincronização e paralelização das tarefas.

No entanto, devido a complexidade do sistema e a falta de tempo hábil, não conseguiu-se implementar utilizando o FreeRTOS, de forma que havia vazamento de memória e as threads não sincronizavam conforme os requisitos de projeto, não obtendo o resultado esperado. No caso do ROS2, por ser um framework mais complexo, não foi possível o estudo aprofundado para conseguir uma implementação que solucionasse o problema.

Logo, a implementação da solução foi utilizando bibliotecas nativas do linux (POSIX) na linguagem C/C++, como o mutex.h, threads.h, sys/ipc.h e sys/sem.h, realizando a sincronização, proteção de condições de corrida e paralelização das tarefas. A implementação das funções em C que descrevem o fluxo do ADCVRT e dos semáforos podem ser vista nas Sessões II.1 e II.2 do apêndice, respectivamente.

3.1 Descrição do sistema

O ADCV-RT irá complementar os trabalhos anteriores de forma a tornar o sistema ADCV um sistema em tempo real, dividindo-se todas as etapas de identificação do sistema em 11 tarefas descritas a seguir:

1. Captura de frame;
2. Detecção do Aruco;
3. Estimação da Matriz de Rotação do Sistema (referencial câmera para o referencial do corpo);
4. Transformação da Matriz de Rotação para o sistema Rodrigues;

5. Transformação do sistema Rodrigues para Euler;
6. Gravar dados de atitude;
7. Gravar frames correspondentes;
8. Enviar dados via TCP;
9. Mostrar frames ao usuário com dados de atitude
10. Interrupção via teclado para parar a emulação do sistema (ESC);
11. Tarefa de Idle para tempos menores que 33ms;

As tarefas 1,6-9 são tarefas em tempo real, ou seja, dependem de interação com o meio e armazenamento de dados. Já as tarefas 2-5 são tarefas não classificadas em tempo real e executadas de forma sequencial. Para o processamento dos frames em 3 eixos, faz-se necessário o uso de multicores, de forma paralelizadas. Caso contrário, o atraso do sistema é muito grande rompendo o ciclo pré estabelecido pelo hardware da câmera de 33ms (30Hz). Dessa forma, a modelagem se dará para apenas a câmera Z, que poderá facilmente ser estendida para a X e a Y, uma vez que os processos são semelhantes.

Modelou-se o projeto como sendo necessário três tipos de tarefas: periódicas, aperiódicas e esporádicas. A Tabela de precedências 3.1 indica todas as tarefas periódicas que necessitam ser executadas anteriormente a tarefa analisada em questão, ou seja, para o término da tarefa 5, necessita-se que a 1, 2, 3, 4 e a 5 sejam executadas. Para a tarefa aperiódicas tem-se a tarefa idle, que é um tempo ocioso do processador aguardando o próximo frame, e para esporádicas tem-se a interrupção do teclado, a qual só é executada após todo o ciclo de processamento de um frame, garantindo que aquele dado seja transmitido para a mesa de testes.

A tarefa de interrupção do teclado não necessita saber o tempo de execução, uma vez acionada, não há mais a preocupação de se perder um frame, o programa sai após ser completada todas as outras tarefas, garantindo-se que a orientação do *Mockup* seja salva no seu arquivo de log. É importante ressaltar que a qualquer momento pode ser pressionada a tecla ESC para parar a emulação, mas esse evento só ocorre após o término de todas as tarefas, vide Tabela 3.1. Já a tarefa de idle, caracterizada como uma tarefa aperiódica, tem seu tempo de execução variado uma vez que é o estado ocioso do processador até obter próximo frame.

Dessa forma, percebe-se que a única tarefa que não tem precedências é a 1 (aquisição de frame), de forma que todas as outras dependem de alguma tarefa anterior e as tarefas 6,7,8 e 9 possuem condição de corrida e não são precedentes umas das outras, sendo possível o escalonamento dessas. Já as tarefas 10 e 11 só podem ocorrer após todas as tarefas serem executadas.

Tabela 3.1: Tabela de precedências do sistema.

As tarefas marcadas como -"não possuem precedências.	
Tarefa	**Precedências**
1	-
2	1-2
3	2-3
4	3-4
5	4-5
6	5-6
7	5-7
8	5-8
9	5-9
10	5-6-7-8-9-10
11	5-6-7-8-9-11

3.2 Simulações no UPPAAL

A fim de definir os tempos para a simulação, mediu-se os tempos de resposta de cada tarefa separadamente, conforme a Tabela 3.2. Assim, utilizou-se a biblioteca chronos em C++ para a medição e um script em python para a análise estatística dos tempos.

Os testes realizados para a validação do projeto utiliza-se de videos capturados em experiemntos realizados no incio de 2020, de forma a se ter 1040 frames, ou seja, cerca de 34320 milissegundos. Por essa amostra mediu-se os dados das propriedades das tarefas presentes na Tabela 3.2, já a análise estatística do sistema se dá apenas no Capítulo 4.

Modelou-se no UPPAAL® o escalonador microkernel com o algoritmo de escalonamento Round-Robin aplicado apenas nas tarefas que não possuem precedência (6 a 9). Vale ressaltar que o UPPAAL é um software apenas para validação lógica do arranjo de tarefas e não para a validação do projeto do sistema em tempo real.

As simulações no software UPPAAL foram realizadas considerando um tempo fixo de 2ms e uma fatia de tempo de 3ms ($20*10^{-1}$ms e $30*10^{-1}$ms na simulação dada pela Figura 3.1), de forma a não infringir as medidas da tarefa 2 na Tabela 3.2. Assim, o estado inicial é a tarefa 1, onde há a aquisição dos frames. Após a tarefa 1 ser completada, identifica-se o padrão de Aruco e armazena-se os pontos que serão utilizados para achar a matriz de rotação entre a câmera e o sistema do corpo, dado pelo Aruco, na tarefa 2.

Tabela 3.2: Dados experimentais medidos a partir de protótipos de funções que descrevem as tarefas em 1040 amostras. As variáveis T (tarefas), D (deadline), C (tempo de execução médio), DP (desvio padrão de C) e P (período) descrevem as características do sistema de acordo com suas tarefas. A tarefa 11 não foi pontuada pois essa será implementada apenas como um delay pelo tempo restante de ciclo.

Características do sistema.					
T	**D (ms)**	**C (ms)**	**DP**	**P (ms)**	**Tempo real**
1	2.500	1.000	0.056	33	Sim
2	3.000	3.000	0.887	33	Não
3	0.500	0.146	0.026	33	Não
4	0.030	0.020	0.001	33	Não
5	0.100	0.060	0.002	33	Não
6	2.800	1.000	0.261	33	Sim
7	2.100	0.030	0.050	33	Sim
8	2.100	0.022	0.058	33	Sim
9	1.000	0.250	0.060	33	Sim
10	33.000	-	-	33	Sim

Computando-se a matriz de rotação na tarefa 3, ela é convertida para os sistemas de referência Rodrigues e Euler nas tarefas 4 e 5, respectivamente. Os pontos da imagem são projetados no mundo real, tendo-se a orientação do satélite com relação ao Aruco no ponto de calibração (0,0,0) de referência do sistema. Após a determinação da atitude do satélite, fecha-se o ciclo sequencial dada na Figura 3.2 caracterizando as tarefas 1-5.

Após a parte sequencial, as tarefas de log (salvar frame e dados de atitude), mostrar para o usuário a representação em Euler e enviar os dados de atitude do ADCV para o *Mockup*, tarefas 6 a 9, são executadas pelo escalonador utilizando-se o algoritmo Round Robin com fatias de tempo de $30*10^{-1}$ms, conforme a Figura 3.3. Essa etapa só é concluida após o término de todas as tarefas, simbolizada pela condição (d5 & d6 & d7 & d8 & d9), onde a variável d* simboliza o estado de tarefa terminada.

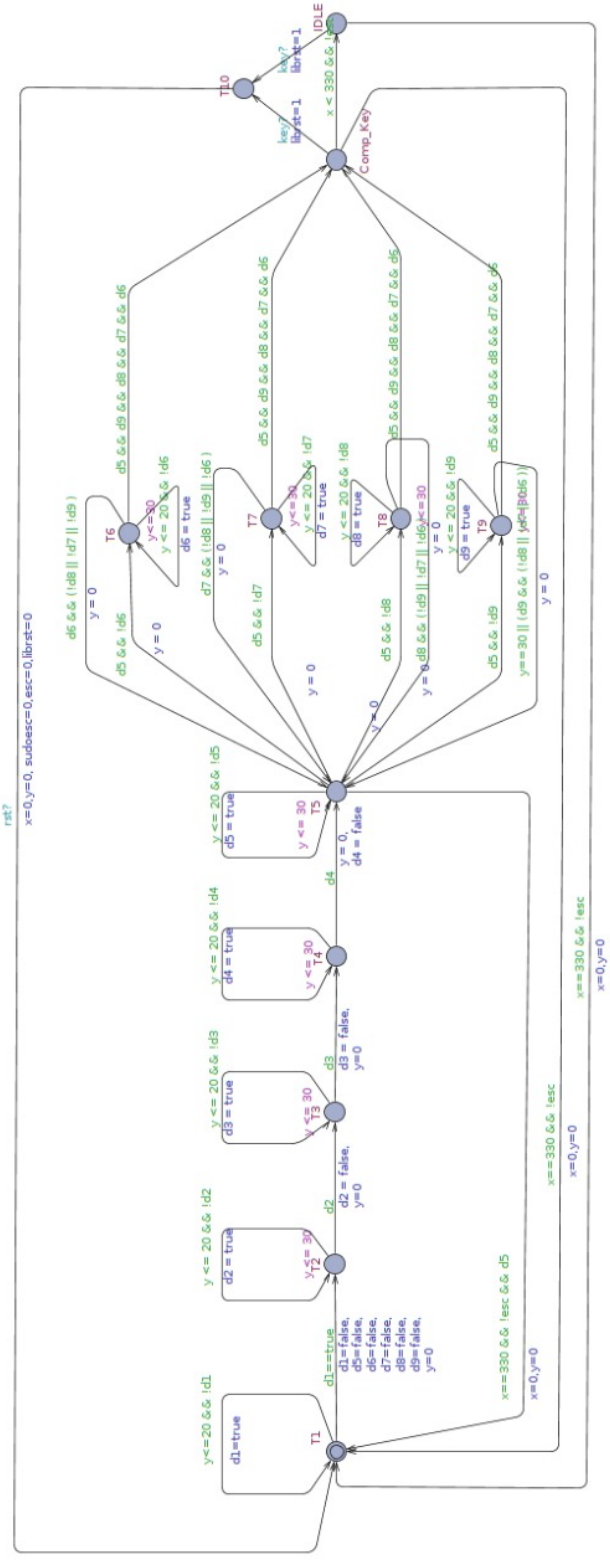


Figura 3.1: Simulação do processo realizada no software UPPAAL.

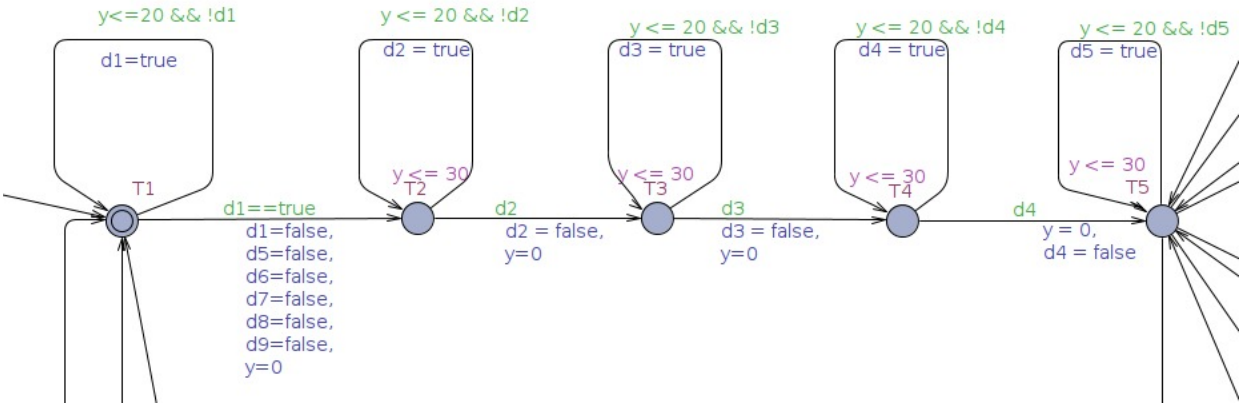


Figura 3.2: Zoom da parte sequencial presente na simulação dada na Figura 3.1.

Em qualquer momento da simulação é possível apertar ESC, ou selecionar a opção **k** para emular a tecla ESC na simulação do UPPAAL, de forma que no estado *comp_key*, que é um condicional no código em C++, há a execução da tarefa 10 (T10), simbolizando o término do programa, como evidenciado na Figura 3.3.

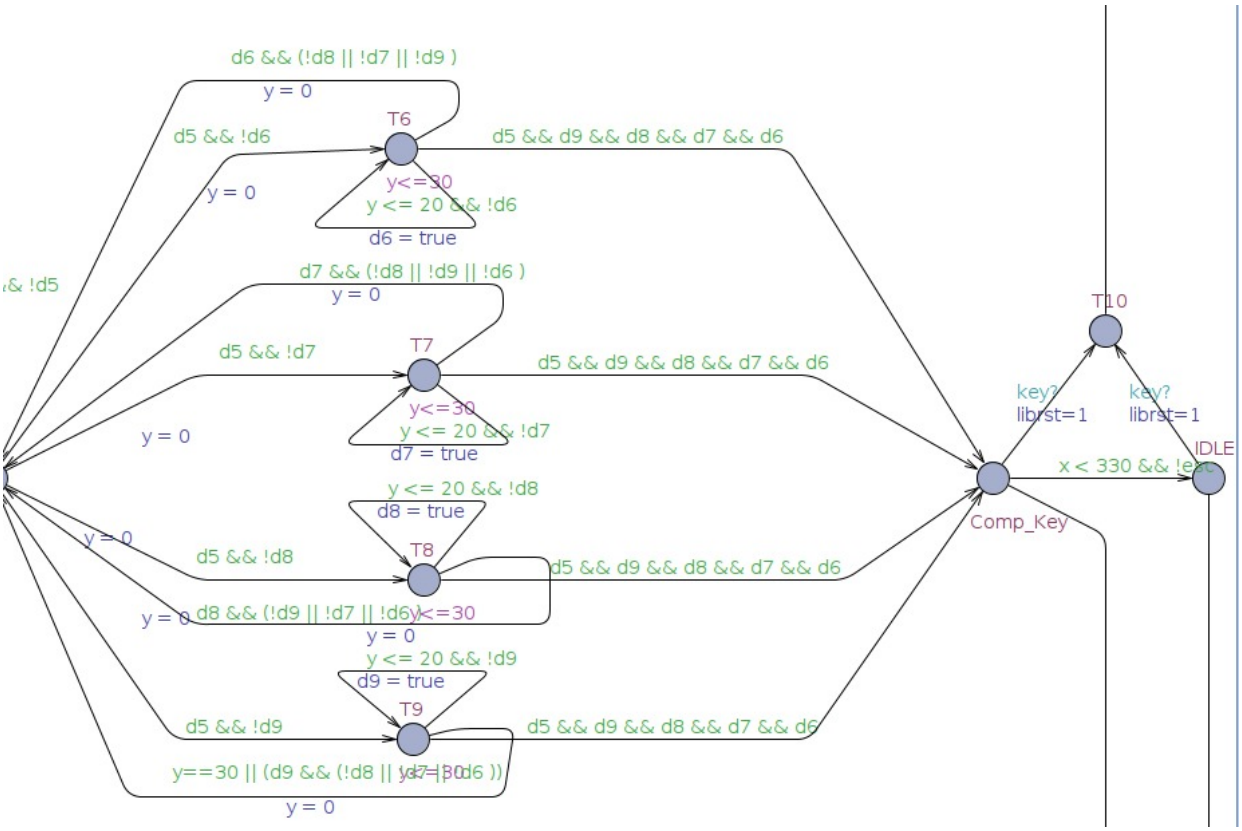


Figura 3.3: Zoom da parte que será escalonada referente a simulação da Figura 3.1.

Uma vez apertado ESC, há a execução da tarefa 10 e o sistema volta para a tarefa 1, com os contatores do loop e as variáveis de estado resetados conforme a Figura 3.1. Esse cenário da tarefa 10 foi implementado apenas em simulação, já que no simulador de pequenos satélites a reinicialização do sistema se dá pelo cancelamento da execução do código.

Caso não tenha sido pressionado ESC (escolhido pela letra **k** durante a simulação), a tarefa 11 é executada, colocando o sistema em estado de idle até o próximo frame (o contador ADCV.x igual a 330), retornando para o início e estabelecendo as condições iniciais de simulação. No caso em que o deadline ($330*10^{-1}$ms) é cumprida ou ultrapassada, retorna-se para T1 em que há uma nova captura de frame. Tal situação pode acontecer após T5 ou após o estado *comp_key*, conforme a Figura 3.1.

Por fim, executou-se a simulação e foi observado que a modelagem cumpre o projeto, sem problemas lógicos de execução.

3.3 Implementação do ADCVRT

Devido a problemas encontrados com o escalonador do FreeRtos e na documentação do Xenomai, optou-se por implementar o sistema com threads e semáforos, para garantir a paralelização das tarefas 6 a 9 e sincronização entre as tarefas dependentes (1 a 5) para cada câmera. O sistema consistem em dois semáforos, *o semáforo do ADCV* para sincronização entre as threads de processamento dos frames das câmeras em 3 eixos (x, y e z) e *o semáforo de dados* que controla a liberação das threads referentes aos dados (tarefas 6 a 9).

Faz-se importante destacar que a condição de bloqueio do semáforo, ou seja, de travamento da thread, se dá quando o seu contador está diferente de zero. Assim, o incremento e decremento desse contador garante a proteção da região de conflito, eliminando as condições de corrida entre threads.

As tarefas 1 a 5 foram implementadas em uma thread por câmera (X, Y e Z), que controlam a do semáforo dos dados, conforme Figura 3.4. Todas as threads são inicializadas simultaneamente, de forma que as threads das tarefas de dado pulam a primeira interação, incrementando o semáforo que fica bloqueado esperando as threads do ADCV finalizar o processamento dos frames, o que é simbolizado pelo estado de início e a sincronização.

No início do loop de processamento do ADCV na Figura 3.5, o semáforo referente às câmeras é incrementado. Após o ADCV (tarefas 1 a 5) processar os frames referente a cada eixo, cada thread decrementa o contador do semáforo, de forma a garantir a sincronização entre essas 3 threads já que o contador volta a zero e há o desbloqueio do semáforo *dados* e o bloqueio do semáforo *ADCV*.

Ou seja, após o desbloqueio, é implementado o estado de idle, que foi incorporado no final do escopo do loop de cada thread como um delay até o ciclo definido de 33ms (30Hz da câmera). Ao passo que o semáforo do ADCV desbloqueia e gera a sincronização, é decrementado o semáforo do DATA que havia sido incrementado em uma etapa anteriro, conforme descrito anteriormente.

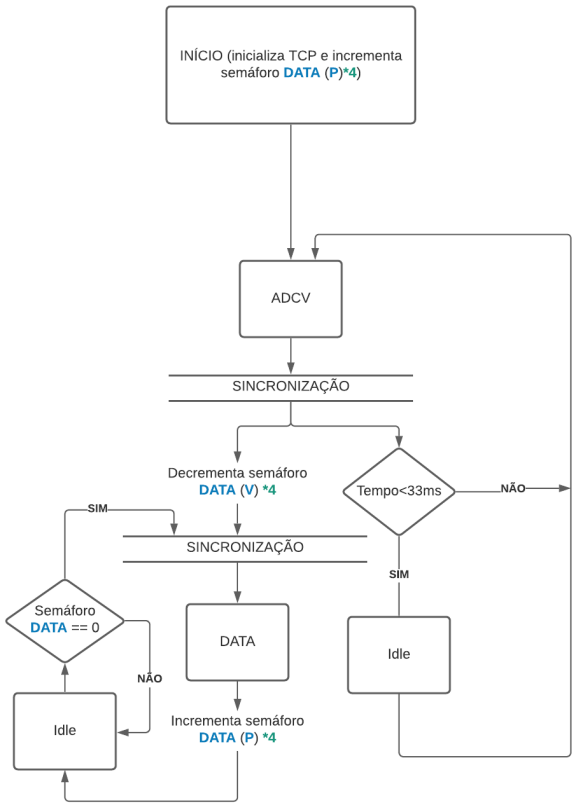


Figura 3.4: Esquemático com uma visão macro do funcionamento dos semáforos e relacionamento entre threads para o ADCV com 3 câmeras.

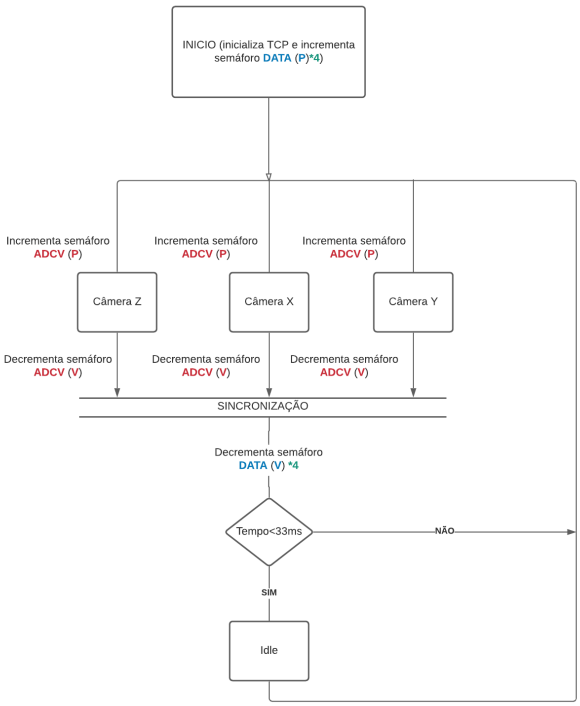


Figura 3.5: Esquemático detalhado das threads e do semáforo correspondente ao processamento dos frames das câmeras X, Y e Z e a determinação de atitude, ou seja as tarefas 1 a 5 para as 3 câmeras. A implementação das funções que descrevem o fluxo dos semáforos pode ser vista na Sessão II.2 do Apêndice.

Uma vez liberado, enquanto as threads do ADCV estão em estado de idle, as threads de dado executam paralelamente, de forma que não causam interferências diretas uma sobre a outra e o fato de ultrapassar o tempo de 33ms relativo ao início do ADCV (captura de frame) passa a não ser mais tão crítico. Esse fluxo lógico é exemplificado na Figura 3.6.

Em sequência, há a execução de cada thread de dado (tarefas 6 a 9), o semáforo é incrementado novamente e cada thread espera no início do seu loop de execução ser desbloqueada pelo semáforo do ADCV, sincronizando-as e garantindo que serão executadas apenas uma vez por frame.

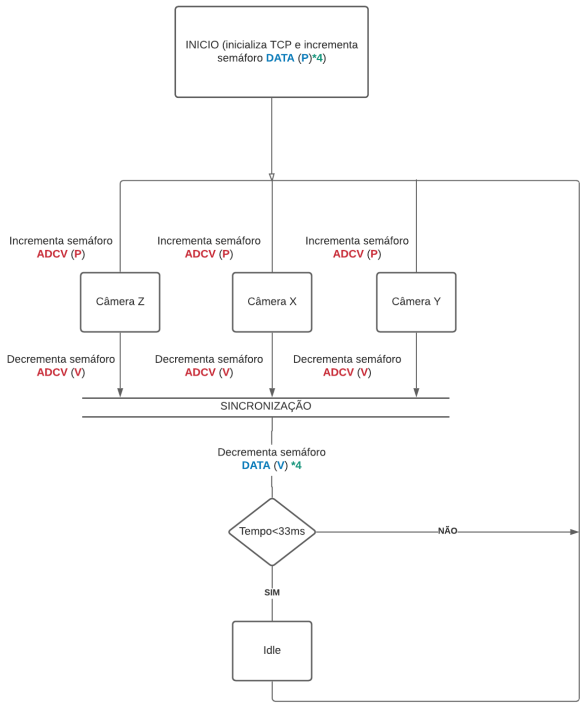


Figura 3.6: Esquemático detalhado das threads e do semáforo correspondente as operações realizadas com os dados resultantes do ADCV, ou seja as tarefas 6 a 9. A implementação das funções que descrevem o fluxo dos semáforos pode ser vista na Sessão II.2 do Apêndice.

Em suma, na primeira interação as threads de dados executam sem realizar nenhuma operação, apenas incrementando o semáforo de dados. Paralelamente, o semáforo do ADCV é incrementado conforme a quantidade de câmeras desejadas na simulação no início das threads das câmeras X, Y e Z, e então são executadas as tarefas 1 a 5 para cada câmera. Após a determinação de atitude, o semáforo é decrementado por cada thread do ADCV que fica bloqueada até todas terminarem, garantindo a sincronização. Após o contador do semáforo DATA chegar em zero, liberando as threads correspondente as tarefas 6 a 9 enquanto o ADCV está em idle.

Com a execução de cada thread, no final de seu loop de execução, o semáforo de dados é incrementado, o que gera o bloqueio para o próximo ciclo, até a liberação pelo semáforo do ADCV, conforme explicitado na Figura 3.4 e detalhado nas Figuras 3.5 e 3.6. Analogamente a Figura 1.13, as threads e a sincronização por semáforo pode ser abstraida no fluxograma representado na Figura 3.7. As threads X, Y e Z são referentes ao semáforo do ADCV e as outras do semáforo de DATA,

sendo essas ocorrendo no ciclo seguinte a aquisição dos frames.

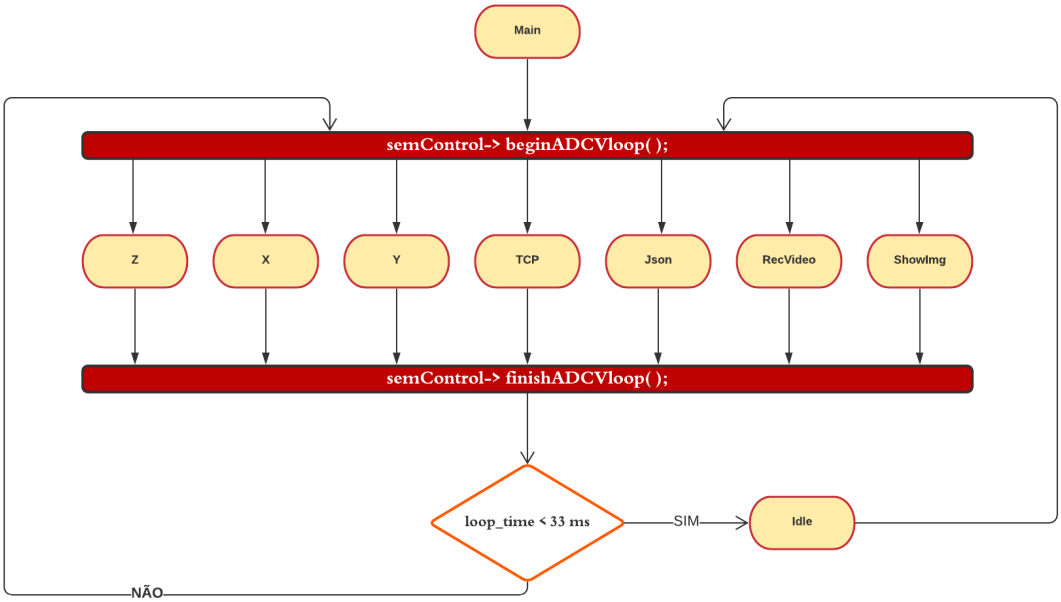


Figura 3.7: Fluxograma do código **main_rts.cpp** com relação a paralelização por threads e sinconização por semáforos. A implementação das funções que descrevem o fluxo do ADCVRT pode ser vista na Sessão II.1 do Apêndice.

Capítulo 4

Análise das Condições de Tempo Real

Como a análise das condições em tempo real dependem diretamente do hardware em que é executado o sistema [7], foram realizados dois testes: no computador de projeto descrito na Subsessão 4.1.2 e no computador do laboratório LODESTAR 4.2.3.

4.1 Análise no Computador de Projeto

Os testes realizados nessa sessão consistem em rodar o ADCVRT com os videos gravados em um experimento previo, de forma a se ter 1040 frames, equivalente a 34.20 segundos de video, como mencionado na Sessão 3.2. Assim, no lugar da aquisição dos frames por meio das câmeras, há a leitura do frame no video gravado. Além disso, por não se dispor de um Raspberry Pi (RPI), foi realizada a comunicação TCP em loopback.

A comunicação em loopback consiste no servidor e cliente estarem rodando na mesma máquina, sem que a comunicação de fato saia para a rede internet e ficando restrita a interface de rede local 127.0.0.1. Isso implica no ADCVRT e o código do RPI rodarem no mesmo computador, ficando isolado do tráfego de rede.

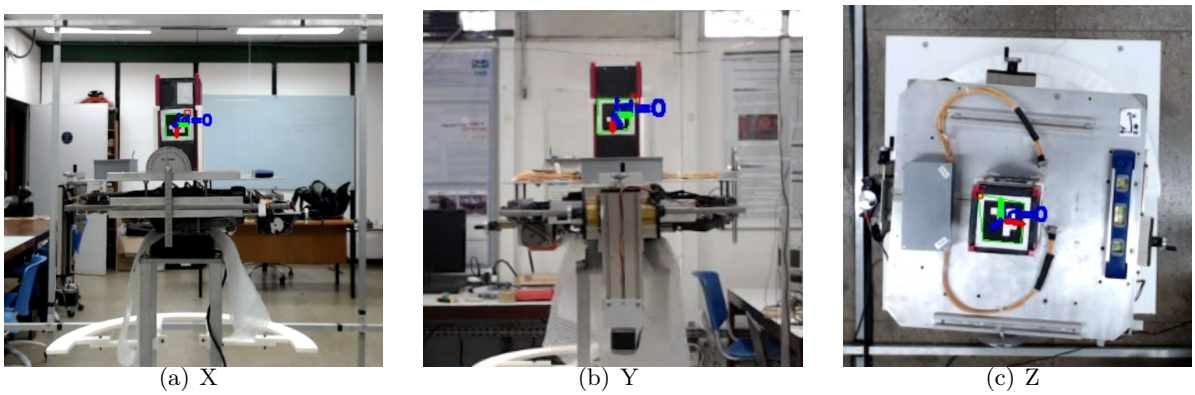
Para tanto, esse teste serve para validar o tempo de resposta das tarefas e a determinação probabilística dos piores casos do tempo de resposta (pWCRT) das tarefas, tendo-se as curvas de distribuição de pareto de todas as tarefas presente no Apêndice I.1. Como no teste realizado não se gravou os dados da IMU, não foi possível realizar as comparações e seguir exatamente o procedimento do Capítulo 5 de análise da atitude no laboratório.

4.1.1 Determinação da Atitude pelo ADCV-RT

O ADCV, sem a interface gráfica, funciona conforme a Figura 4.1, onde se tem o *Cubesat* 2U no centro da mesa instrumentada, o qual possui os padrões de Aruco presos a sua fuselagem. Os

eixos em vermelho, verde e azul simbolizam a atitude, ou seja, os ângulos de orientação X, Y e Z na representação de Euler.

Na versão mais atualizada, a fuselagem do *Cubesat* 2U foi trocada a fim de garantir maior precisão quanto ao alinhamento do padrão de Aruco, como será mostrado nos capítulos seguintes, e o alinhamento dos transferidores será explicado no Capítulo 5.



(a) X (b) Y (c) Z

Figura 4.1: Imagens do ADCV após a determinação de atitude para os ângulos [10.50, -9.32, 82.23] no referencial de Euler.

4.1.2 Testes dos Tempos de Resposta e de Ciclo do ADCV-RT

Nos testes preliminares serão utilizados vídeos obtidos pelas câmeras em testes realizados no laboratório no início do ano de 2020, com cerca de 34320 milissegundos de duração, arquivos de log e, a princípio, uma comunicação TCP em *loopback* para emular a troca de dados entre o computador Desktop (estação solo) e a MCU da mesa de testes. O computador utilizado para avaliar os testes possuem as características de hardware e SO: intel i9-9700k, 16Gb de Ram 3666Hz, uma placa de video 1660 e placa mãe Asus 350m TUF-gaming, utilizando-se o sistema operacional Ubuntu 20.

Foram realizados alguns testes em códigos iniciais para elaborar a simulação no UPPAAL medindo-se o tempo de execução por meio da biblioteca *chrono* e obtendo-se média e desvio padrão das tarefas para os 1040 frames gravados, conforme a Tabela 3.2 apresentada no Capítulo 3. Em seguida procedeu-se com a implementação do sistema em tempo real, explicitado na Subsessão 3.3.

Foram colhidas as amostras de tempo das tarefas, optando-se por uma abordagem POT (*Peak Over Threshold*) explicada na Sessão 2.5.5.2 com TH igual média mais uma unidade de desvio padrão dos dados amostrados. Essa métrica de um desvio padrão acima da média foi escolhida devido a quantidade de 1040 amostras. Em seguida, obteu-se as curvas PDF, CDF e 1-CDF da generalizada de Pareto.

Colheram-se dados considerando as tarefas de 1 a 5 como uma única tarefa, sendo implementada uma Thread para cada câmera X, Y e Z. Quanto as outras tarefas, os dados foram tratados

separadamente. Com os tempos colhidos, foi usado um programa em python para obter os seguintes dados estatísticos: média, mediana, desvio padrão, variância, mínimo observado, máximo observado(HWM) e a probabilidade de ocorrência do HWM (pHWM). Nas Tabelas 4.1, 4.2 se encontram os dados colhidos, onde a estimação obtida (pWCRT) é o HWM acrescido de uma margem de segurança de 20% [7].

Tabela 4.1: Dados referentes às tasks 1-5(ADCV) das câmeras X, Y e Z.

DATA.								
Câmera	**Média**	**Mediana**	**Desvia Padrão**	**Variância**	**Min**	**Max(HWM)**	**p(HWM)**	**pWCRT**
X	6.63	7.65	1.91	3.64	3.9	4.19	0.02%	16.476
Y	6.49	05.02	2.21	4.89	3.3	12.15	0.03%	14.58
Z	7.53	6.71	2.35	5.53	3.75	19.25	0.01%	23.1

Tabela 4.2: Dados referentes às tasks 6-9(DATA).

ADCV.								
Task	**Média**	**Mediana**	**Desvia Padrão**	**Variância**	**Min**	**Max(HWM)**	**p(HWM)**	**pWCRT**
6	0.04	0.03	0.06	3	1	0.99	0.1%	1.19
7	16.8	16.63	1.12	1.26	14.81	31.31	7%	37.57
8	0.04	0.03	0.02	6.1e-4	4.1e-3	0.53	0.02%	0.64
9	0.67	0.62	0.48	0.23	0.38	8.95	0.01%	10.74

Além disso, obtiveram-se os gráficos das funções PDF, CDF e 1-CDF da Generalizada de Pareto. As Figuras 4.2, 4.3 e 4.4 exemplificam as curvas obtidas para as câmeras X, Y e Z. Os gráficos referentes a todas as tarefas modeladas na Tabela 3.1 estão todas localizadas no Apêndice I. Para a PDF estimada, dada a natureza dos dados, definiu-se empiricamente que as variáveis μ, σ e ζ são equivalentes a 1, 1000 e a média dos dados após ser aplicado o POT.

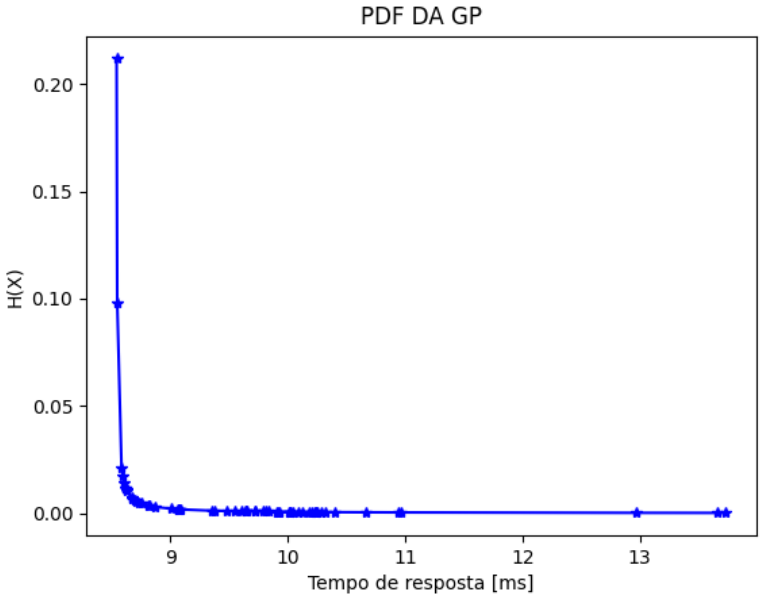


Figura 4.2: Função PDF das tasks 1 a 5 do ADCV da câmera X com relação ao tempo de execução. O último ponto da curva, ou seja, o HWM do sistema é 13.734ms com probabilidade de ocorrência de 0.00019086% para as 1040 amostras.

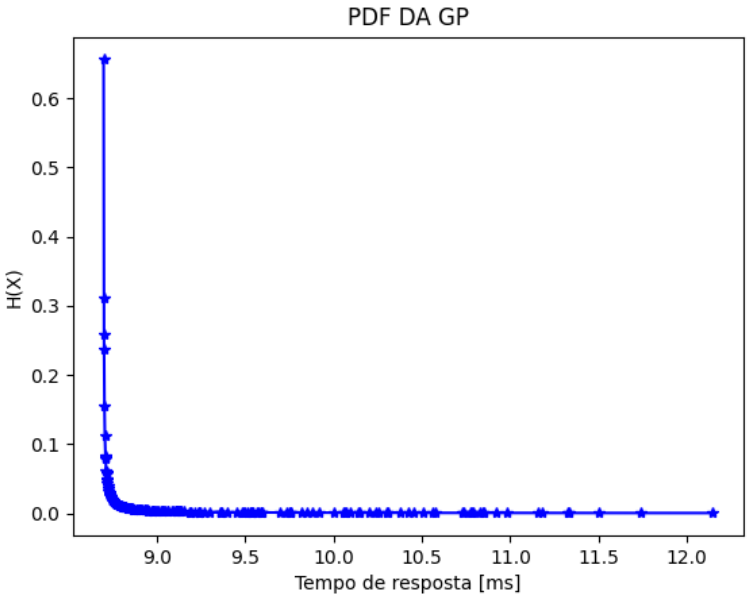


Figura 4.3: Função PDF das tasks 1 a 5 do ADCV da câmera Y com relação ao tempo de execução. O último ponto da curva, ou seja, o HWM do sistema é 12.147ms com probabilidade de ocorrência de 0.00028745% para as 1040 amostras.

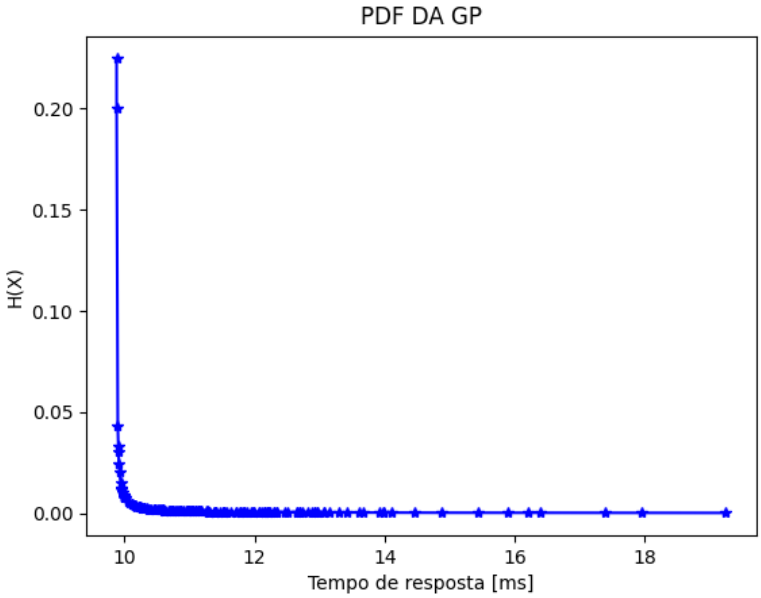


Figura 4.4: Função PDF das tasks 1 a 5 do ADCV da câmera Z com relação ao tempo de execução. O último ponto da curva, ou seja, o HWM do sistema é 19.253ms com probabilidade de ocorrência de 0.00010567% para as 1040 amostras.

4.1.3 Testes para determinação do atraso do TCP em Loopback

Essa análise não consiste no atraso real que o protocolo TCP implica no sistema, mas sim uma análise do tempo de resposta da tarefa de transmissão dos dados de atitude do ADCVRT. Assim, repetiu-se a execução do ADCVRT 5 vezes, medindo-se apenas o tempo de resposta dessa tarefa, com os 1040 frames do experimento.

Como o tempo de resposta depende da situação do sistema operacional percebe-se oscilações nos dados obtidos na Tabela 4.3. Ou seja, a quantidade de cache disponível, uso de processador com outras tarefas que não relacionadas ao ADCVRT, predição de saltos (branch predictor) [7], interferem na resposta da tarefa, mesmo que executando-a com maior prioridade pelo comando *sudo renice -n -19 -p <número_do_processo_de_execução_do_programa_na_saída_do_comando_ps-faux>*.

Pelos dados da Tabela 4.3, percebe-se que a média dos tempos de execução corresponde a 6.75ms e do pWCRT equivalente a 23.10ms. Como ocorre em atraso e o tempo médio de resposta da câmera Z é de 7.53ms com um pWCRT de 23.10ms, que é a câmera que tem o maior tempo de resposta e por isso o fator determinante no funcionamento do semáforo, sendo essa a que desbloqueia o semáforo DATA na maioria das vezes, percebe-se que no pior cenário, o dado de atitude seria enviado via TCP com um atraso máximo de 46.20ms, ou seja, em um ciclo menor que 2 frames (66ms) e sem atrapalhar o dado do próximo ciclo de processamento, por funcionar de forma paralela e síncrona.

Tabela 4.3: Dados do tempo de atraso do TCP.

TCP.								
Teste	**Média**	**Mediana**	**Desvia Padrão**	**Variância**	**Min**	**Max(HWM)**	**p(HWM)**	**pWCRT**
1	6.63	7.65	1.91	3.64	3.9	4.19	0.02%	16.476
2	6.49	5.02	2.21	4.89	3.3	12.15	0.03%	14.580
3	7.53	6.71	2.35	5.53	3.75	19.25	0.01%	23.100
4	6.63	7.65	1.91	3.64	3.9	4.19	0.02%	16.476
5	6.49	5.02	2.21	4.89	3.3	12.15	0.03%	14.580

Dessa forma, no teste realizado em loopback e no computador de projeto, a arquitetura até então escolhida atende a deadline de 33ms de ciclo, pWCRT do TCP igual a 23.10ms, com uma folga ainda para eventuais atrasos não modeláveis [7]. Dada a dinâmica lenta do simulador advindo da contraposição entre a inércia e a potência gerada pelas rodas de reação ou magnetorques, as condições obtidas são razoáveis para o ADCVRT.

4.2 Análise de Desempenho do ADCVRT no LODESTAR

Os testes realizados no laboratório LODESTAR no início de 2021 tem caracter de contrapor os resultados da Sessão 4.1 nas condições em loopback com a comunicação ADCVRT-RPI. Além disso, acentuar a diferença de performance e a variação dos tempos de execução entre os computadores de projeto e o do laboratório.

Quanto aos procedimentos realizados nesse capítulo, apenas diferem da sessão anterior quanto a comunicação do ADCVRT-RPI, onde foi fixado os IPs do computador do ADCVRT e do RPI no roteador encontrado no laboratório a fim de não se ter problemas de conflito de IP entre um experimento e outro. Além disso, ajustou-se o IP da variável *const char* ADCVRT_IP* de 127.0.0.1 para 192.168.0.102 na porta 7000, e executou-se o programa main_controller.c no RPI ao invés de no computador que roda o ADCVRT, que é a definição de comunicação em loopback explicada na Sessão 4.1.

No entanto, devido as características de hardware do computador presente no laboratório, que são inferiores aos do computador de projeto, aumentou-se o tempo de projeto para 3 frames, de 66ms para 100ms. No caso, a etapa de processamento dos frames demora mais de 33ms, colocando-se 2 ciclos pra essa parte e o restante de tempo para as outras tarefas. Outro ponto a se destacar é que os dados coletados de resposta foram adquiridos nas duas pontas da comunicação, tanto do lado do ADCVRT quanto do RPI, sem considerar individualmente cada tarefa como realizado na Sessão 4.1, a fim de se facilitar a determinação do atraso da comunicação e do sistema como um todo. A implementação das funções que originam os gráficos e a análise dos dados pode ser vista na Sessão II.3 do apêndice.

4.2.1 Análise do tempo de resposta - pWCRT

O teste realizado no laboratório LODESTAR, consistiu em executar o ADCVRT nos cenários de comunicação ADCVRT-loopback e ADCVRT-RPI utilizando-se as câmeras X, Y e Z presentes no laboratório, conforme apresentado na Tabela 4.4. Os dados medidos são apenas referentes a tarefa de comunicação TCP, medidos do lado do ADCVRT e do RPI. Percebe-se que a comunicação TCP quase não influencia no tempo médio de reposta, mas quanto ao pWCRT, última coluna da Tabela 4.4, pode ser tornar algo crítico quando mais dispositivos estiverem conectados na rede wifi.

Tabela 4.4: Dados do tempo de atraso do ADCVRT-RPI.

IMU e ADCV.									
Teste	**N_Amostras**	**Média**	**Mediana**	**Desvia Padrão**	**Variância**	**Min**	**Max(HWM)**	**p(HWM)**	**pWCRT**
RPI	2624	37.106	33.809	12.422	154.306	20.022	79.832	0.002%	95.798
ADCVRT	2624	36.411	34.994	8.346	69.656	20.010	73.860	0.003%	88.632
RPI_{lb}	4267	33.558	33.301	5.258	27.647	20.036	72.254	0.003%	86.905
$ADCVRT_{lb}$	4267	33.208	32.966	5.195	26.988	20.033	72.358	0.002%	86.829

Outro ponto a se destacar é o desvio padrão do conjunto de dados para todos os testes realizados, que, quando comparados com o cenário exposto na Tabela 4.3, torna-se evidente a discrepância do tempo de reposta quanto ao hardware utilizado. Em outras palavras, dado o fato do computador do laboratório ser mais antigo e com menos recursos de hardware que o computador de projeto, percebe-se maior oscilação no tempo de resposta [7], mesmo utilizando o comando para aumento da prioridade do ADCVRT.

4.2.2 Comparação entre os Testes Realizados em Loopback e com o RPI

Nos cenários de teste supracitados, pode-se comparar os dois testes (loopback e ADCVRT-RPI) e os seus subsistemas conforme o tempo médio de resposta, a mediana do conjunto de dados, o HMW (máximo do conjunto de dados) e o pWCRT na Figura 4.5. Dessa forma, percebe-se que o atraso médio do TCP, que seria a diferença entre os tempos médios do RPI para o ADCVRT nos casos em loopback e ADCV-RPI, se dá de forma 0.350ms e 2.602ms; e o pWCRT se dá de 0.076ms para 7.166ms, respectivamente.

Além disso, a quantidade de amostras acima das marcas de 33ms e 66ms, conforme mostrado na Tabela 4.5, aumentou consideravelmente para o caso de 66ms, sendo necessário o reajuste do ciclo de tempo de amostragem do ADCVRT e consequentemente o tempo de discretização dos controladores, por exemplo, para 100ms.

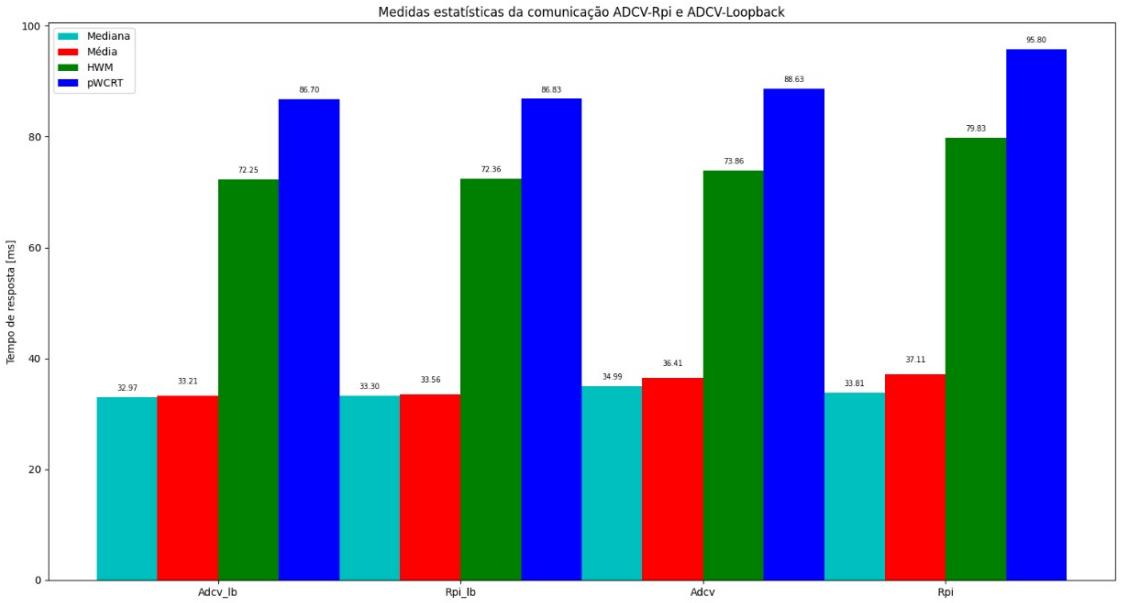


Figura 4.5: Comparação das métricas adotadas para os tempos de resposta do ADCV e RPI para os testes de comunicação do ADCV-loopback e ADCV-RPI.

Tabela 4.5: Porcentagem de amostras acima do tempo de amostragem dos frames para 1 frame (33ms) e 2 frames (66ms) no laboratório LODESTAR.

IMU e ADCV.				
Sistema	**Total**	**Amostras{t} > 33ms**	**Amostras{t} > 66ms**	**N_Amostras{t} > 66ms**
RPI	2624	53.544%	3.877%	106
ADCV	2624	62.500%	0.269%	9
RPI_{lb}	4267	52.332%	0.047%	2
$ADCV_{lb}$	4267	49.637%	0.070%	3

Pela Tabela 4.5, percebe-se que para o tempo de ciclo do RPI_{lb} em comparação com o tempo do RPI, que são os tempos críticos para os algoritmos discretos embarcados, precisa-se ajustar o tempo de processamento dos frames de 1 para 2 frames a fim de não se ter perdas significativas. Para tanto, o tempo de ciclo do algoritmo no RPI deve ser ajustado de 66ms para, aproximadamente, 100ms, a fim de se perder apenas aproximadamente 4% dos frames.

4.2.3 Comparação entre os dois Hardwares

Quando comparado os valores médios e os pWCRTs no computador de projeto, vide Tabela 4.3, percebe-se uma diferença significante, de forma a se ter um pWCRT de 23.10ms, 86.705ms e 95.798ms para os casos do computador de projeto, ADCVRT-loopback e ADCVRT-RPI, respec-

tivamente. Tal fato justifica o acréscimo de 2 frames para 3 frames como ciclo do programa.

Tabela 4.6: Dados do tempo de atraso do ADCV e RPI para os casos do computador de projeto e do computador do laboratório LODESTAR.

IMU e ADCV.			
Teste	**N_Amostras**	**Tempo Médio**	**pWCRT**
RPI_{lab}	2624	37.106	95.798
$ADCVRT_{lab}$	2624	36.411	88.632
RPI_{lb_lab}	4267	33.558	86.705
$ADCVRT_{lb\_lab}$	4267	33.208	86.829
$ADCV_{lb\_projeto}$	1040	6.75	23.10

No entanto, como a quantidade de dados foi menor e o fato de se capturar os frames de um video gravado ao invés dos sensores, os quais possuem um comportamento estocástico, pode influenciar nos tempos de resposta medidos no computador de teste, acrescentando alguns milisegundos a ultima linha da Tabela 4.6. Vale ressaltar que era experado tal diferença nos tempos de execução pelo fato do computador do laboratório ser mais antigo que o computador de projeto, possuindo um procesador i7 da 5 geração, 8Gb de ram e sem placa de video.

Os gráficos das distribuições de pareto de cada tarefa podem ser comparados nos Apêndices I.1 e I.2 do Apêndice. Tendo-se os gráficos referentes ao experimento com o computador do laboratório LODESTAR nas figuras presentes no Apêndice I.2.

Capítulo 5

Análise da Acurácia do ADCVRT

Essa análise consiste no erro estático do ADCVRT com relação a referência estabelecida com uso dos transferidores, um de metal e um impresso em 3D, como na Figura 5.1. O transferidor impresso 3D é graduado de 1 em 1 grau de +90° a -90°, já o transferidor de ferro é graduado de 1 em 1 grau de +45° a -45°.

Dessa forma, os transferidor impresso em 3D foi preso no suporte do mancal a ar e com o auxílio de um apontador a laser (fixado sobre a massa móvel Z) consegue-se ter a componente Z de atitude da mesa, e portanto, a do *Cubesat* 2U, já que pode-se considerar um corpo rígido todo o sistema. Para os eixos X e Y, utiliza-se o transferidor pendular de metal, fixando-o na lateral da mesa e a gravidade inclina a haste do pêndulo aferindo a inclinação da componente analisada.

Fixou-se a mesa sobre o mancal a ar com o uso de um plástico bolha, de forma a ficar estática como na figura 5.2, assim, ajustou-se a sua atitude manualmente seguindo os ângulos nos transferidores e o medidor de nível preso sobre a mesa. Assim, executou-se os sistema ADCVRT e do programa que roda no RPI (*main_controller.c*), coletando-se os dados de atitude do ADCVRT e da IMU. A necessidade do uso do ADCVRT, como explicado anteriormente, se dá pelo fato da IMU perder a referência devido a indução do campo magnético gerado pelas bobinas da gaiola de Helmholtz. A implementação das funções que originam os gráficos e a análise dos dados pode ser vista na Sessão II.4 do Apêndice.

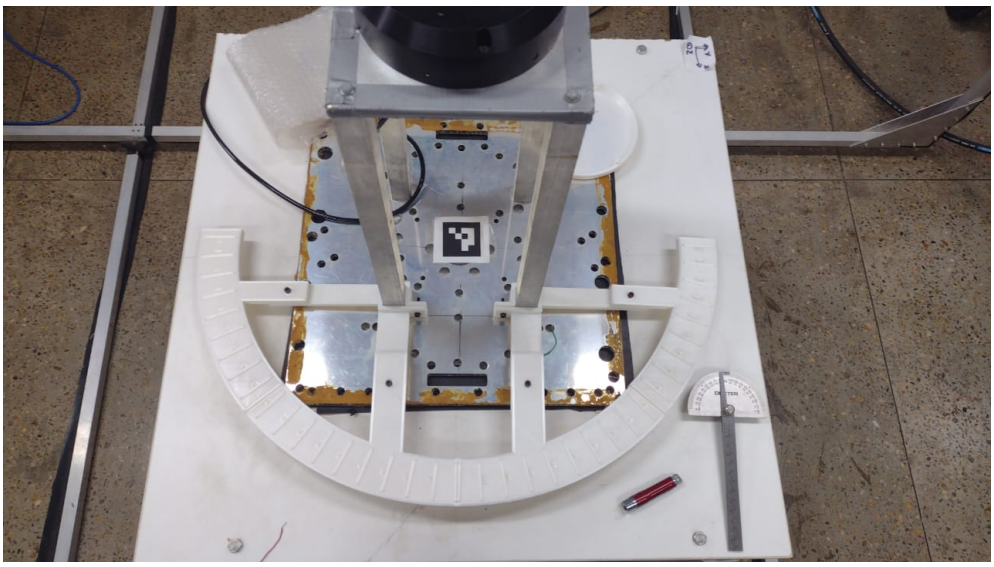


Figura 5.1: Figura representativa dos transferidores: impresso em 3D (branco) graduado a cada 1° de +90° a -90°; e o tranferidor pendular de metal (cinza) ao lado do apontador a laser, graduado a cada 1° de +45° a -45°.

5.1 Teste estático para determinação do erro de Atitude dos ângulos X, Y e Z em Euler do ADCVRT

A partir da coleta dos dados e do ajuste manual do apontamento da mesa instrumentada, percebeu-se que a IMU estava mais próxima dos ângulos de referência adquiridos com os transferidores, de forma que o vetor de referência obtido se aproxima do vetor $\{x, y, z\} = \{7, 0.5, 0\}$, lembrando que no ciclo trignométrico 360° equivale a 0°. Assim, no teste realizado, obteve-se as distribuições estatísticas conforme a Tabela 5.1, onde a oscilação das componentes da IMU tendem a zero e ADCVRT são diferente de zero. Vale ressaltar que a oscilação da atitude da câmera se dá a questões de imprecisão da câmera, como as distorções radiais e translacionais da imagem, e de iluminação do ambiente.

Tabela 5.1: Comparação dos ângulos em euler da IMU e do ADCV em 2624 amostras para a referência $\{x, y, z\} = \{7, 0.5, 0\}$.

IMU e ADCV.						
Componente	**Média**	**Mediana**	**Desvio Padrão**	**Variância**	**Min**	**Max**
$ADCV_Z$	1.345	1.356	0.085	0.007	0.428	1.515
$ADCV_Y$	-3.061	-1.965	2.388	5.702	-7.363	-1.038
$ADCV_X$	-8.574	-8.774	0.453	0.206	-9.191	-4.083
RPI_Z	359.938	359.938	$5.684 * 10^{-14}$	$3.231 * 10^{-27}$	359.938	359.938
RPI_Y	0.502	0.5	0.011	10^{-4}	0.500	0.562
RPI_X	7.000	7.000	0	0	7.000	7.000

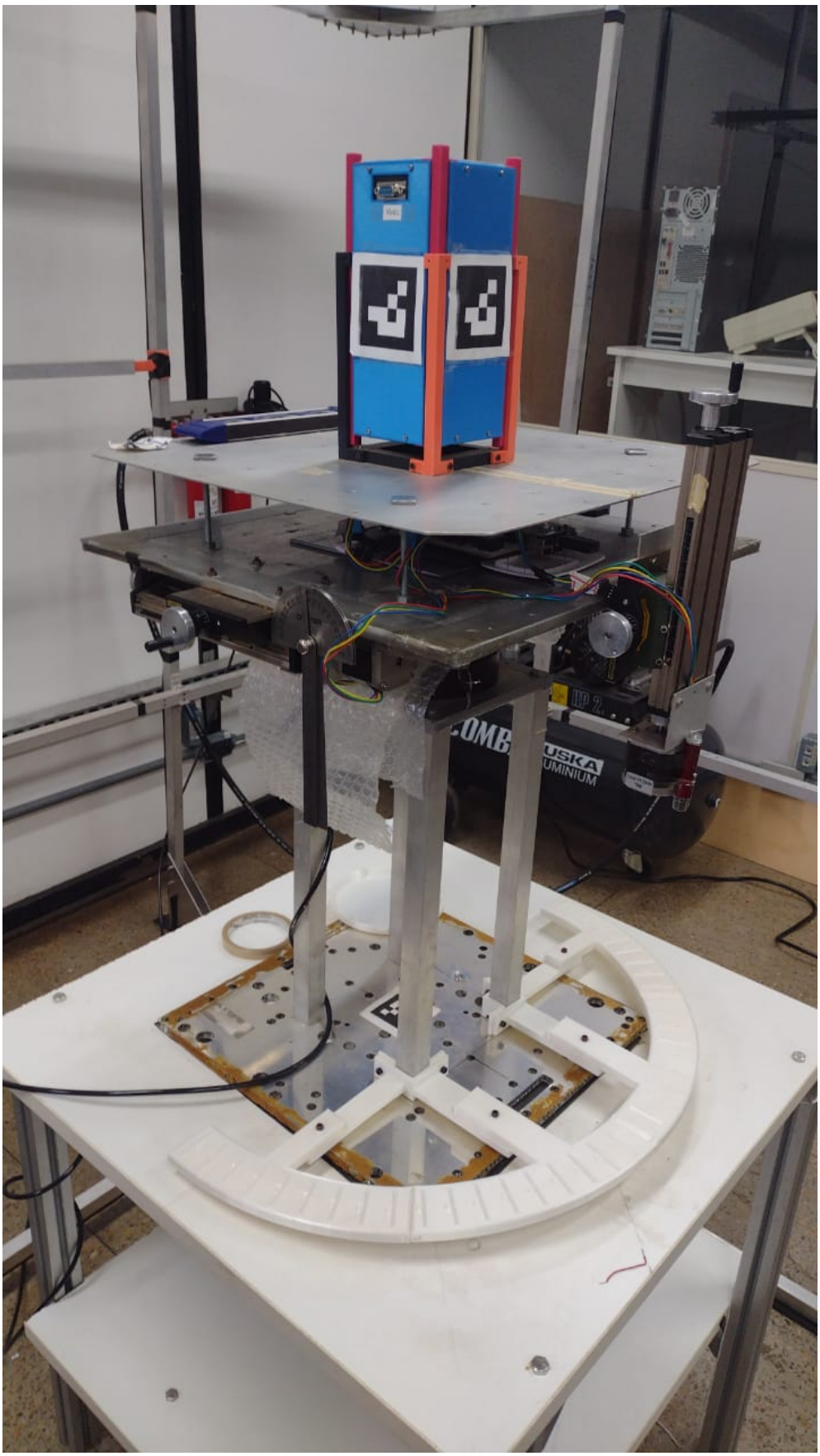


Figura 5.2: Figura representativa do *Mockup* 2U - *Cubesat* de testes e não qualificado para voô - sobre a mesa instrumentada e o mancal a ar. Exemplo de fixação dos transferidores impressos em 3D e de pendular de metal com o apontador a laser fixado na massa móvel Z.

Pela Tabela 5.1 percebe-se que a maior imprecisão do ADCVRT se dá na componente Y, onde a câmera fica fisicamente posicionada a favor de uma janela no laboratório. Devido à variação da iluminação externa e incidência de luz, o que provoca a saturação e sombras no padrão de Aruco, o padrão é mais dificilmente detectado de forma a se ter um desvio padrão maior que as outras componentes STD({x,y,z}) = {0.453, 2.388, 0.085}. Os valores médios e a mediana referentes a cada componente dos dois sensores representam apenas a disposição física dos padrões de Aruco sobre o *Cubesat* 2U, ou seja, um erro grosseiro experimental que pode ser ajustado para todas as referências terem os os valores mais próximos possível.

Assim, como o referencial da IMU (BFF) praticamente está alinhada ao referencial dos transferidores, que são o da gaiola (ORB), a IMU será considerada como a referencia para a análise dos erros do ADCVRT. Sendo assim, computou-se os offsets, os erros quadráticos médios, a raíz não normalizada e normalizada do erro quadrático médio na Tabela 5.2, que são medidas de erro comumente aplicadas para o sensor da IMU com relação ao referencial fixado. O offset computado se deu pela menor diferença absoluta entre os ângulos dentro do ciclo trigonométrico e o NRMSE, que geralmente está entre 0 e 1, ultrapassou esse limite devido ao fato do mínimo e máximo estarem muito próximos, com uma diferença próxima de 1 e a diferença entre a medida e a referência é maior que 1.

Tabela 5.2: Cálculo dos offsets do ADCV em relação a IMU para 2624 amostras.

IMU e ADCV.					
Componente	**Offset**	**Erro_STD**	**MSE**	**RMSE**	**NRMSE**
Z	1.408	0.085	1.286	1.655	1.522
Y	3.563	2.388	2.388	5.701	0.901
X	15.574	0.453	1.638	2.684	0.525

Pela Tabela 5.1, consegue-se determinar uma equação de calibração do ADCVRT com relação a IMU, onde a componente linear se daria pelo valor médio ou a mediana e a ângular pela regressão das curvas com o ângulo desejado e a entrada da IMU, como mostrado nas equações representadas na Equação (5.1).

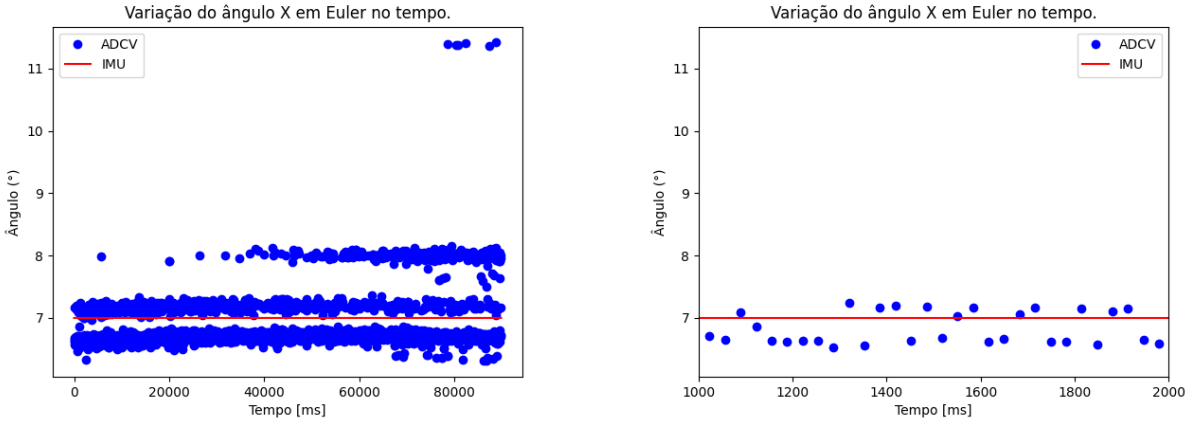
$$\begin{aligned} F_X(\phi) &= 0.184 * \phi + 8.574; \\ F_Y(\alpha) &= -5.102 * \alpha + 3.061; \\ F_Z(\theta) &= 1.004 * \theta - 1.345. \end{aligned} \tag{5.1}$$

5.2 Comportamento temporal da atitude do ADCVRT

Devido a variação de iluminação e as imprecisões dos cálculos matriciais ocasionados por hardware, percebe-se que há uma oscilação das componetes da matriz de atitude determinada pelo ADCVRT dada na Tabela 5.1. Dessa forma, pretende-se mostrar como a evolução temporal

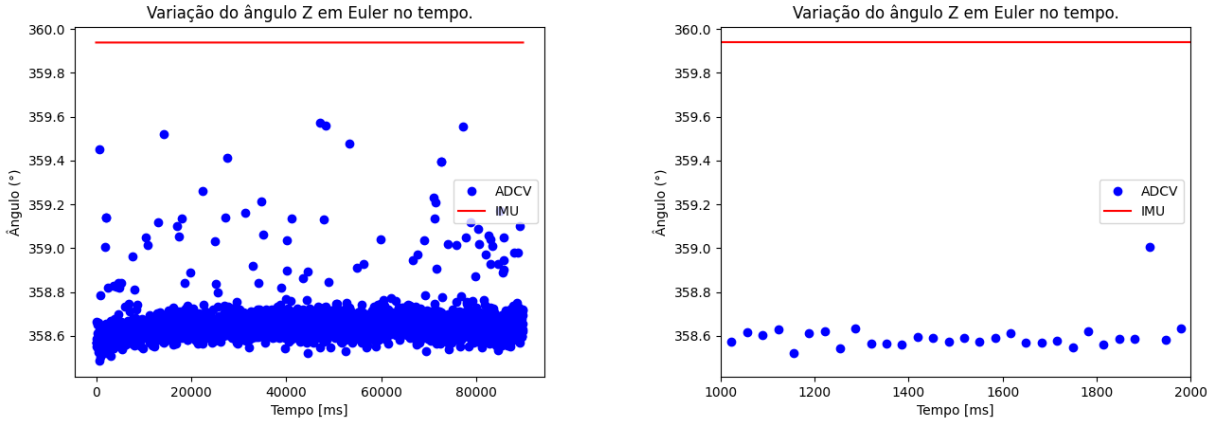
do teste influencia na determinação de atitude por visão computacional, tendo-se a IMU plotada como a referência do sistema.

Assim, nos gráficos das Figuras 5.3, 5.5 e 5.4, percebe-se a distribuição dos dados de atitude determinados pelo ADCVRT em azul no padrão de bolas e a linha contínua em vermelho as leituras da IMU. É notório que nos gráficos presentes nas Figuras 5.3 e, principalmente, 5.4 a grande maioria dos dados encontram-se perto da referência, justificada pelo baixo desvio padrão. Vale ressaltar que para o gráfico da componente X, as entradas foram multiplicadas por -1, uma vez que apenas denota o sentido do padrão Aruco, mas não de fato a sua orientação.



(a) Dataset Completo (b) Zoom

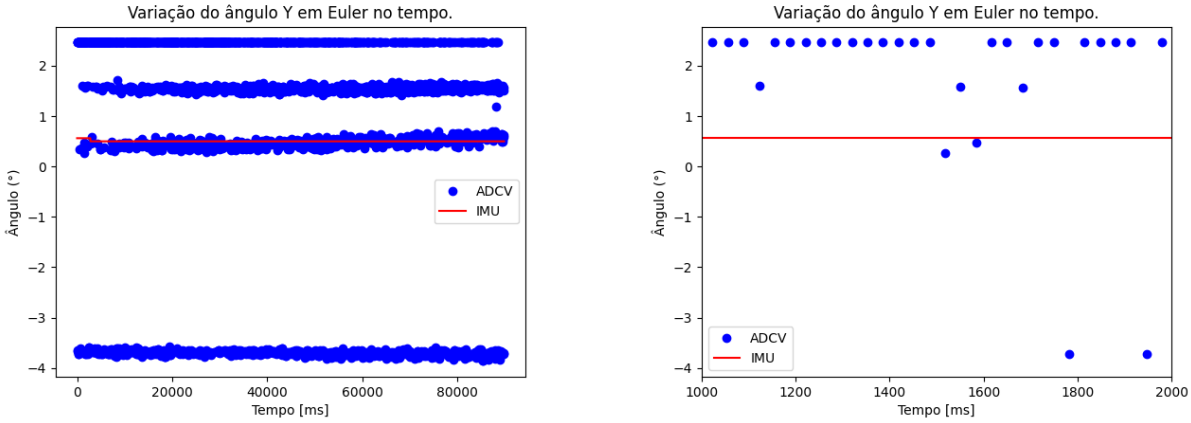
Figura 5.3: Distribuição dos dados coletados de atitude da câmera X com o tempo. Na Figura 5.3(a) é representado todo o conjunto de dados ao longo do tempo do esperimento em milisegundos. Já na Figura 5.3(b) é representada apenas um subconjunto dos dados coletados.



(a) Dataset Completo (b) Zoom

Figura 5.4: Distribuição dos dados coletados de atitude da câmera Z com o tempo. Na Figura 5.4(a) é representado todo o conjunto de dados ao longo do tempo do experimento em milisegundos. Já na Figura 5.4(b) é representada apenas um subconjunto dos dados coletados.

No que tange a componente Y, dado o gráfico da Figura 5.5, percebe-se que a dispersão temporal segue um padrão determinado pelas 4 "retas"em azul formadas pelos dados e é acentuado no zoom representado pela Figura 5.5(b). Tal característica, como levantado anteriormente na hipótese do erro dessa componente, pode estar atrelado a iluminação e na dificuldade da detecção dos pontos do Aruco, o que é caracteriza o ruido acrescido na câmera Y. Uma abordagem para eliminação dessas interferencias, poderia ser a alteração do espaço físico, de forma que se controlasse melhor a iluminação interna do laboratório aliado ao uso de um filtro de kalman na saída do ADCVRT, de forma que as componentes seriam filtradas e tais oscilações seriam reduzidas.

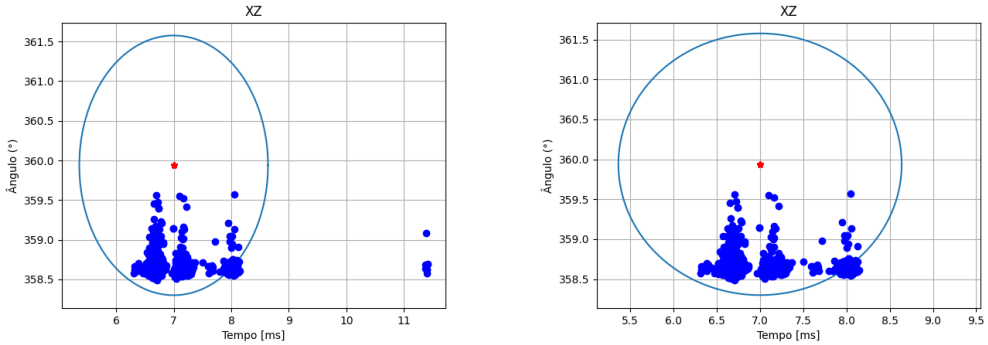


(a) Dataset Completo (b) Zoom

Figura 5.5: Distribuição dos dados coletados de atitude da câmera Y com o tempo. Na figura 5.5(a) é representado todo o conjunto de dados ao longo do tempo do esperimento em milisegundos. Já na Figura 5.5(b) é representada apenas um subconjunto dos dados coletados.

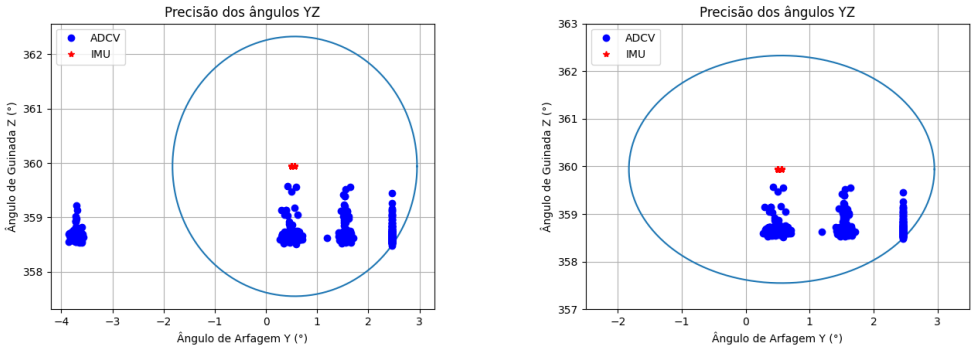
Além disso, representou-se a relação de acurácia e precisão da atitude no plano determinada pelo ADCVRT por meio dos gráficos das Figuras 5.6, 5.7 e 5.8. Assim, o raio plotado representa erro quadrático médio entre as componentes de atitude coletado da IMU e determinado pelo ADCVRT, ou seja, o maior MSE dentre as componentes analisadas, que foram calculadas na Tabela 5.2, por exemplo: entre as componentes XY, foi pego o maior MSE que está na componente Y.

Para tanto, percebe-se que o erro no plano é mais considerável com as combinações que consideram a componente Y. Dado os indicadores gerados, justifica-se os erros atrelados aos planos formados com essa componente, já que foi a que obteve as maiores discrepâncias quanto ao referencial. No entanto, é notório que, de modo geral, os erros de apontamento do sistema não são críticos dependendo das aplicações ou simulações, mas devem ser minimizados a ponto de se ter maior consistência nos dados, ou seja, um menor desvio padrão tanto para a determinação de atitude quanto para as condições de tempo real do ADCVRT.



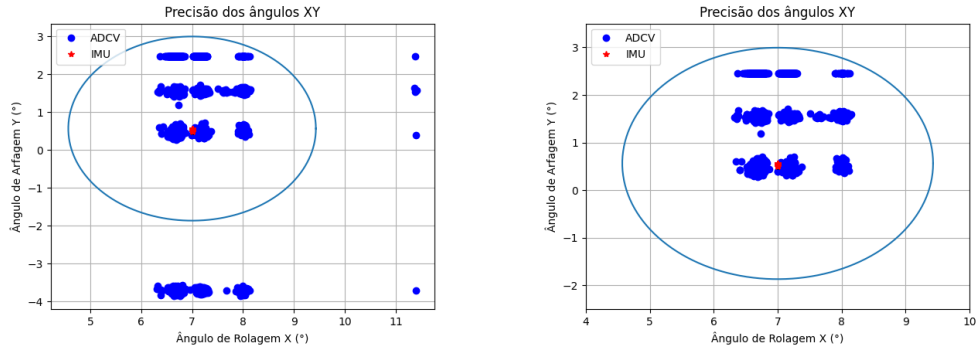
(a) Dataset Completo (b) Zoom

Figura 5.6: Gráfico de dispersão de atitude no plano XY com um raio de 1.286. A Figura 5.6(b) representa o zoom de todo o dataset na Figura 5.6(a).



(a) Dataset Completo (b) Zoom

Figura 5.7: Gráfico de dispersão de atitude no plano YZ com um raio de 2.389. A Figura 5.7(b) representa o zoom de todo o dataset na Figura 5.7(a).



(a) Dataset Completo (b) Zoom

Figura 5.8: Gráfico de dispersão de atitude no plano XY com um raio de 2.389. A Figura 5.8(b) representa o zoom de todo o dataset na Figura 5.8(a).

Capítulo 6

Conclusões

Neste trabalho realizou-se a continuação do aprimoramento da mesa instrumentada quanto ao software dela e do simulador de pequenos satélites localizado na unidade de processamento central (estação solo). Além disso, projetou-se o sistema em tempo real por visão computacional ADCVRT, objetivando-se o maior desempenho e modularização dentro da arquitetura de software do simulador de pequenos satélites, e realizou-se testes para a estimação das variáveis de erro, atraso do sistema e acurácia do ADCVRT em dois cenários de teste: computador de projeto e na unidade de processamento central no LODESTAR.

Como visto no Capítulo 4, percebe-se que o hardware, mais especificamente as características internas dos processadores, utilizados no laboratório e o de projeto é um fator determinístico para o bom funcionamento do ADCVRT, em vista que aplicações de visão computacional e determinação de atitude possuem uma carga de trabalho grande e alto custo computacional devido às operações matriciais. Mais fatores que corroboram pra diferentes tempos de resposta das tarefas pode ser encontrado na referência [7].

Vale ressaltar que devido ao fato das condições de projeto não terem ficado em 33ms para a captação e processamento dos frames e 33ms para o envio dos dados via TCP nos testes realizados no laboratório, que eram as condições de projeto, recomenda-se utilizar o ADCVRT com um ciclo de programa de 100ms, devido aos resultados obtidos. Assim, a taxa de discretização do sistema deve ser ajustada para 100ms. No entanto, devido ao tempo de processamento demorar dois frames nos piores casos analisados (cerca de 66ms), perde-se um frame na amostragem do sensor, podendo ter problemas no funcionamento da câmera e perdendo-se um pouco da confiabilidade do projeto para sistemas de dinâmica rápida. Não obstante, esse fenômeno não se aplica ao simulador de pequenos satélites já que a sua dinâmica de rotação é lenta.

6.1 Perspectivas Futuras

Além das melhorias sugeridas, como mudanças no ambiente do laboratório LODESTAR a fim de se garantir maior uniformidade da luz e reajuste dos padrões de Aruco para anular os offsets calculados, recomenda-se a implementação de uma GUI que não atrapalhe o ADCVRT quanto

ao seu ciclo de funcionamento, ou seja, um programa que comunique em loopback enviando as imagens e os dados de atitude lidos da IMU e do ADCVRT. A implementação da biblioteca genérica de controladores seguindo os padrões de projeto expostos durante o trabalho e a realização do mesmo teste em diferentes condições de ângulo para a determinação de equações de calibração mais abrangentes.

Segundo as recomendações da banca, em casos em que a comunicação TCP se torne crítica na análise de tempo de resposta da comunicação, poderia-se implementar as seguintes soluções: uso de uma VPN *site to site* conectando diretamente a unidade central de processamento no Raspberry Pi com o uso do protocolo UDP, assim teria-se o fator de segurança na comunicação aliada a velocidade de transmissão do UDP em comparação ao TCP/IP. Outro método seria a utilização do protocolo UDP com o uso de algorítmos hashing para garantir uma segurança na transmissão de dados junto com um mecanismo de verificação de mensagem enviada, análogo ao protocolo TCP/IP. Por fim, sugeriu-se a implementação de outros métodos de determinação de atitude por visão computacional, como o método de detecção dos pontos de malha do cubesat, juntamente com os testes para análise de desempenho do sistema de visão computacional.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] ANDRADE, L. *Proposta de Sistema de Determinação de Atitude por Visão Computacional em Simulador de Pequenos Satélites*. https://conferencias.unb.br/index.php/iniciacaocientifica/25CICUnB16df/paper/view/20619. Acessado: 07-09-2020.

[2] SILVA, R. C. da. *Simulador de Sistema de Determinação e Controle de Atitude de Pequenos Satélites*. 2015. Tese de conclusão de curso na Universidade de Brasília. Brasília, Brazil.

[3] SILVA, R. C. da. *Filtering and Adaptative Control for Balancing a Nanosatellite Testbed*. Dissertação (Mestrado) — Universidade de Brasília, Brasília, Brasil, 2018.

[4] VIEIRA, M. *Proposta de Sistema Embarcado Eletrônico para Simulador de Pequenos Satélites*. https://conferencias.unb.br/index.php/iniciacaocientifica/26CICUNB17DF/paper/view/34178. Acessado: 27-03-2021.

[5] GUIMARÃES, F. C. *Implementation of Attitude Determination Techniques for a Small Satellite Three-axis Simulator*. Dissertação (Mestrado) — Universidade de Brasília, Brasília, Brasil, 2018.

[6] SILVA, F. A. e. *c-tcp-helper-lib*. https://gitlab.com/flavioasilva2/c-tcp-helper-lib. Acessado: 07-09-2020.

[7] OLIVEIRA, R. S. de. *Fundamentos dos Sistemas de Tempo Real*. [S.l.]: Amazon, 2018. (SPACE TECHNOLOGY LIBRARY, v. 1). ISBN 13/9781728694047.

[8] SILVA, R. C. da et al. Helmholtz cage design and validation for nanosatellites hwil testing. *IEEE Transactions on Aerospace and Electronic Systems*, IEEE, 2019.

[9] KINOSHITA, I. S.; SILVA, L. M. B. da. *Teste de um Algorítmo para Controle Magnético da Atitude de um Satélite*. 2018. Tese de conclusão de curso na Universidade de Brasília. Brasília, Brazil.

[10] ISHIOKA, I. S. K. et al. Hil Testing of the B-dot Attitude Control Law. *Proceedings of the III IAA Latin American Cubesat Workshop, Ubatuba*, v. 11, p. 1–9, 2018.

[11] MODENINI, D. et al. A dynamic testbed for nanosatellites attitude verification. *Aerospace*, v. 7, p. 31, 03 2020.

[12] PLOEG, L. C. van der. *Desenvolvimento de um Sistema para Simulação do Campo Magnético em Órbitas Baixas.* 2017. Tese de conclusão de curso na Universidade de Brasília. Brasília, Brazil.

[13] LOIOLA, J. V. L. et al. 3 Axis Simulator of the Earth Magnetic Field. *Proceedings of the 39th IEEE Aerospace Conference*, p. 1–8, 2018.

[14] SILVA, R. C. et al. Tabletop Testbed for Attitude Determination and Control of Nanosatellites. *Journal of Aerospace Engineering*, v. 32, p. 1–9, 2019.

[15] ANDRADE, L. *Proposta de Arquitetura de Software para Sistema em Simulador de Pequenos Satélites Baseado em IoT.* https://conferencias.unb.br/index.php/iniciacaocientifica/25CICUnB16df/paper/view/20619. Acessado: 07-09-2020.

[16] ANDRADE, L.; SILVA, F. *ADCV-RPI System Integration.* https://gitlab.com/hil-table-nanosatelites-simulator/adcv_rpi. Acessado: 07-09-2020.

[17] MARKLEY, F. L.; CRASSIDIS, J. L. *Fundamentals of Spacecraft Attitude Determination and Control.* [S.l.]: Microcosm Press and Springer, 2014. (SPACE TECHNOLOGY LIBRARY, v. 1).

[18] OPENCV.ORG. *Tutorial de calibração de câmeras.* https://docs.opencv.org/4.5.1/d9/db7/tutorial_py_table_of_contents_calib3d.html. Acessado: 12-05-2021.

[19] HIBBELER, R. C. *Engineering Mechanics Dynamics.* [S.l.]: Pearson Education, 2010. (Dinâmica: Mecânica para Engenharia, v. 12).

[20] YOUNG, J. S. *Development of an Automatic Balancing System for a Small Satellite Attitude Control Simulator.* Dissertação (Mestrado) — PhD thesis, Utah State University. Department of Mechanical and Aerospace Engineering., Utah, Estados Unidos, 1998.

[21] YANG, Y. *Spacecraft Modeling, Attitude Determination, and Control: Quaternion-Based Approach.* [S.l.]: CRC Press, 2019. (SPACE TECHNOLOGY LIBRARY, v. 1).

[22] SSH.COM. *SFTP Protocol.* https://www.ssh.com/ssh/sftp/. Acessado: 27-03-2020.

ANEXOS

I. GRÁFICOS E SIMULAÇÕES

I.1 Gráficos PDF, CDF e 1-CDF Realizados no Computador de Projeto

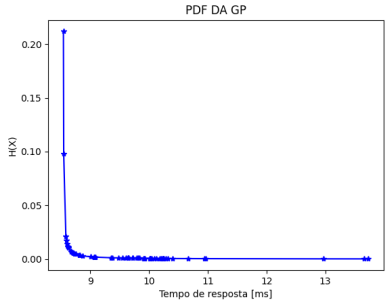


Figura I.1: Função PDF das tasks 1 a 5 do ADCV da câmera X

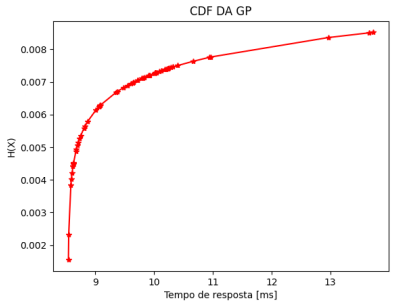


Figura I.2: Função CDF das tasks 1 a 5 do ADCV da câmera X

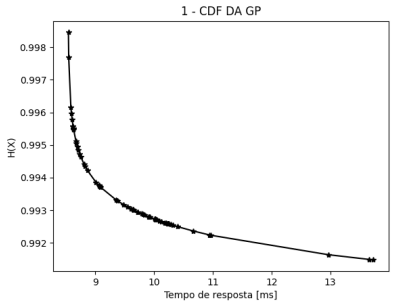


Figura I.3: Função 1-CDF das tasks 1 a 5 do ADCV da câmera X

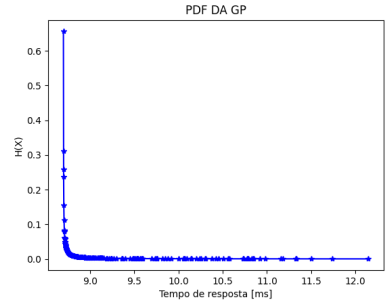


Figura I.4: Função PDF das tasks 1 a 5 do ADCV da câmera Y

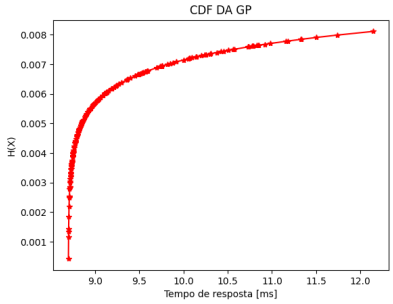


Figura I.5: Função CDF das tasks 1 a 5 do ADCV da câmera Y

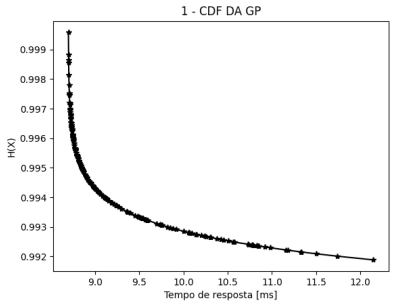


Figura I.6: Função 1-CDF das tasks 1 a 5 do ADCV da câmera Y

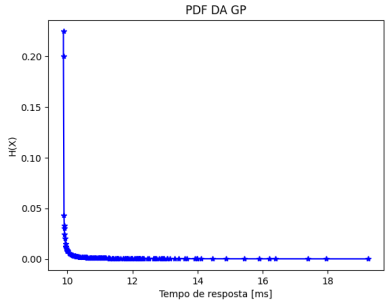


Figura I.7: Função PDF das tasks 1 a 5 do ADCV da câmera Z

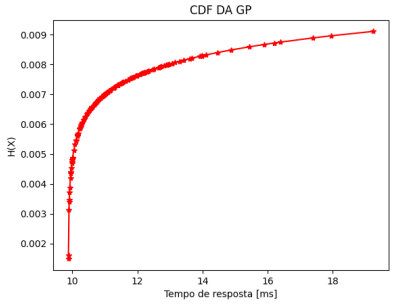


Figura I.8: Função CDF das tasks 1 a 5 do ADCV da câmera Z

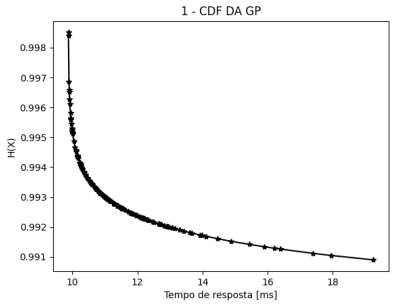


Figura I.9: Função 1-CDF das tasks 1 a 5 do ADCV da câmera Z

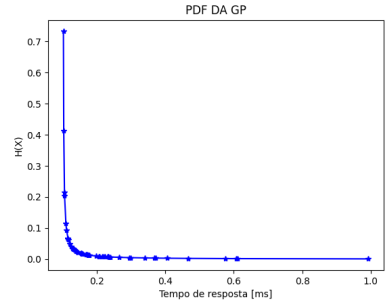


Figura I.10: Função PDF da task 6

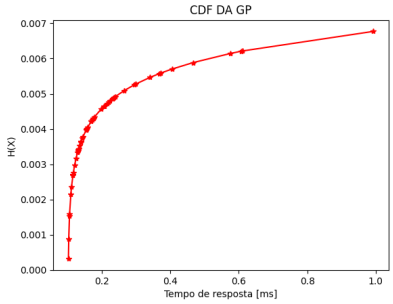


Figura I.11: Função CDF da task 6

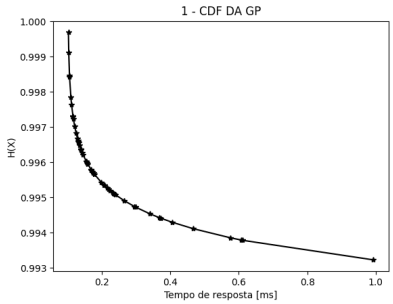


Figura I.12: Função 1-CDF da task 6

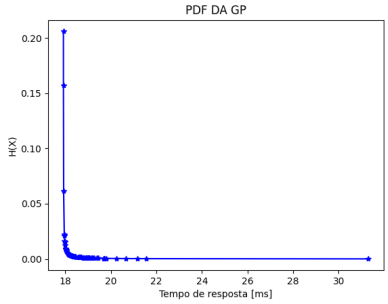


Figura I.13: Função PDF da task 7

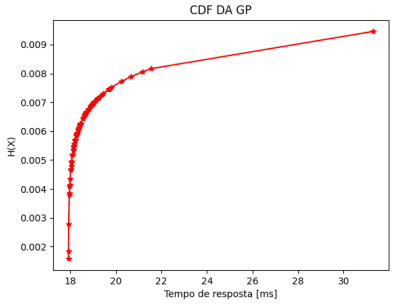


Figura I.14: Função CDF da task 7

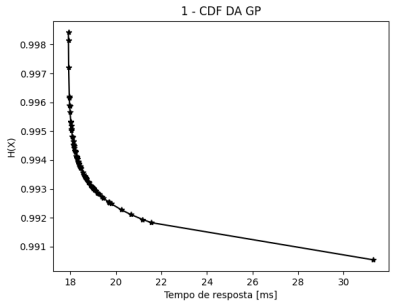


Figura I.15: Função 1-CDF da task 7

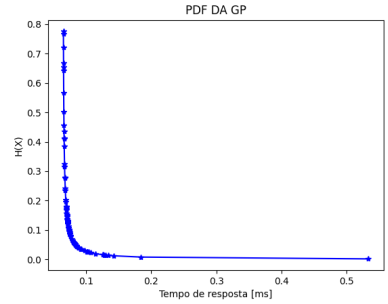


Figura I.16: Função PDF da task 8

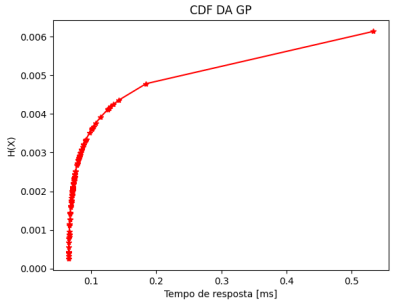


Figura I.17: Função CDF da task 8

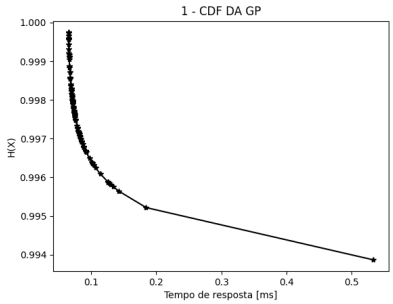


Figura I.18: Função 1-CDF da task 8

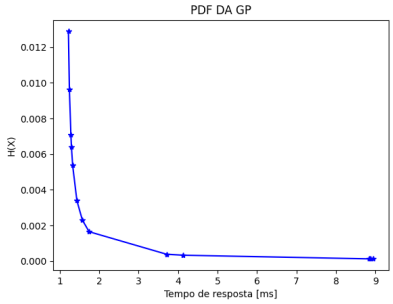


Figura I.19: Função PDF da task 9

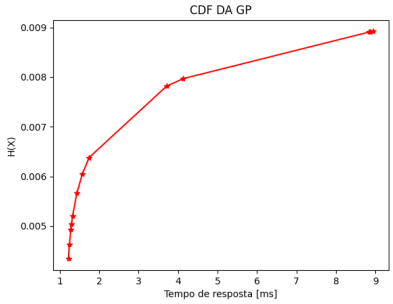


Figura I.20: Função CDF da task 9

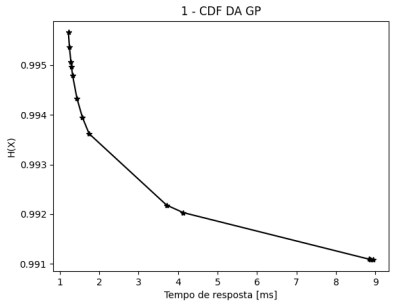


Figura I.21: Função 1-CDF da task 9

I.2 Gráficos PDF, CDF e 1-CDF Realizados no Computador do LODESTAR

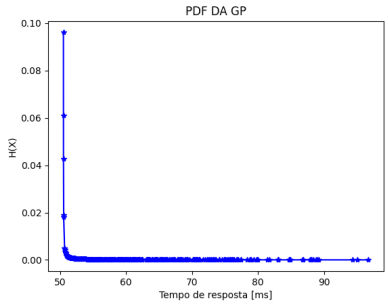


Figura I.22: Função PDF do RPI

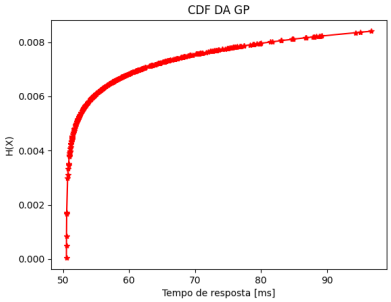


Figura I.23: Função CDF do RPI

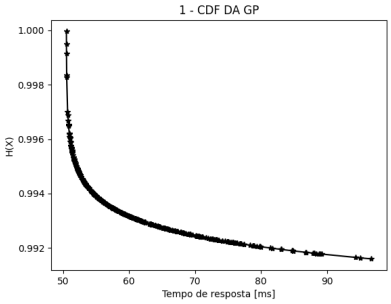


Figura I.24: Função 1-CDF do RPI

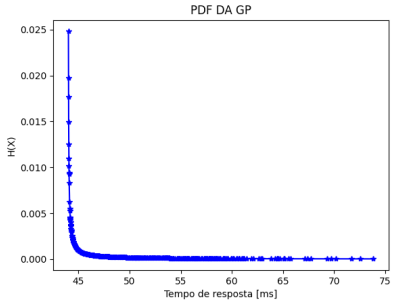


Figura I.25: Função PDF do ADCV

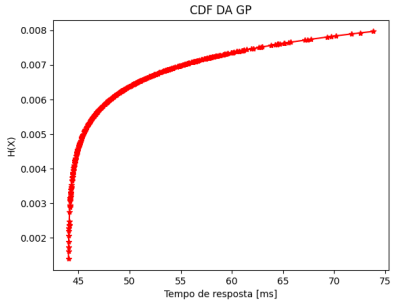


Figura I.26: Função CDF do ADCV

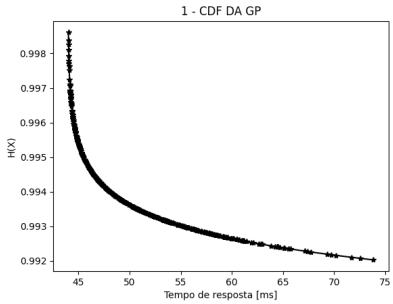


Figura I.27: Função 1-CDF do ADCV

II. ROTINAS DE TESTE

II.1 Exemplo de código main do ADCVRT

```
#include <adcv.hpp>
#include <server.hpp>
#include <iostream>
#include <string>
#include <chrono>
#include <exitCodes.hpp>
#include <fstream>
// Paralellization libs
#include <thread>
#include <pthread.h>
#include <errno.h>
#include <string.h>
#include <tcpdata.h>
#include <signal.h>
#include <semaphores.hpp>
#include <jsonLogging.hpp>

using namespace std;

typedef struct ARGS{
    bool z               = false;
    bool all             = false;
    bool rec             = false;
    bool video           = false;
    bool paralellization = false;
    string z_video       = "";
    string y_video       = "";
    string x_video       = "";
}ARGS;

#define BASE_DIR_RTS "time_rts/"
#define DETERMINE_RTS_TIME 1

#define CAMERA_FRAME_RATE 30
#define ESC_KEY 27 // Olhar no doc da opencv

int globalSemId = 0;
const double CYCLE_TIME = 1000/CAMERA_FRAME_RATE; //(ms)

char c = '\0'; // Key to quit
const unsigned short int SERVER_PORT = 7000;


void argumentParser(ARGS *args, int argc, const char** argv);
```

```
45 void initTCPServerConnection(TCPServer **srv, unsigned short port);
46 Connection* getConnection(TCPServer **srv);
47 void adcvLoop(ADCV *adcv, int cam, Semaphores *semControl); // Tasks 1,2,3,4,5
48 // void adcvLoop(ADCV *adcv, int cam, Semaphores *semControl, JSONLogging *log,
       Connection *con); // Tasks 1,2,3,4,5
49 void dataLog(ADCV *adcv, Semaphores *semControl, JSONLogging *log, int numCam);
       // Task 6
50 void recVideo(ADCV *adcv, Semaphores *semControl); // Task 7
51 void tcpLoop(ADCV *adcv, Semaphores *semControl, Connection *con, int numCam); //
        Task 8
52 void showImg(ADCV *adcv, Semaphores *semControl); // Task 9
53
54 void removeSemaphores(int signum);
55
56 int main(int argc, const char** argv){
57     /**
58      * Argparse:
59      *
60      *  -a -p -r -v
61         @args.z
62         @args.all
63         @args.rec
64         @args.video
65         @args.z_video
66         @args.y_video
67         @args.x_video
68     */
69
70     ARGS args;
71     argumentParser(&args, argc, argv);
72
73     Semaphores semSet = Semaphores();
74     globalSemId = semSet.getADCVSemId();
75     signal(SIGINT, removeSemaphores);
76     signal(SIGTERM, removeSemaphores);
77
78     JSONLogging logInstance;
79     /**
80      * éMtodos opcionais:
81      *
82      * logInstance
83         .setFilePath("arquivoDeLog.json")
84         .setAuthorName("Lukas Lorenz de Andrade")
85         .setTestLocal("LODESTAR")
86         .setProjectName("Testbed integration");
87     */
88
89     ADCV *adcv;
90     if(args.video)
91         adcv = new ADCV(args.x_video, args.y_video, args.z_video, args.rec);
92     else
93         adcv = new ADCV(args.all, args.rec);
```

```
94
95      // Starting simulation:
96      TCPServer* srv = NULL;
97      Connection* con = NULL;
98
99      cout << "Waiting for TCP ADCV-RPI TestBed Connection at port: " <<
        SERVER_PORT << endl << endl;
100     initTCPServerConnection(&srv, SERVER_PORT);
101     con = getConnection(&srv);
102
103     cout << "TCP ADCV-RPI Connected!" << endl;
104     cout << "Starting Simulation!" << endl;
105     Semaphores semaphores = Semaphores();
106     semaphores.setQuitConstant(ESC_KEY);
107     // thread adcvThreadX, adcvThreadY;
108     thread adcvThreadX, adcvThreadY, tcpThread, showThread, dataThread, recThread
        ;
109     // Cameras
110     if(args.all) {
111         adcvThreadX = thread(adcvLoop, adcv, CAMX_ID, &semaphores);
112         pthread_setname_np(adcvThreadX.native_handle(), "adcvThreadX");
113         adcvThreadY = thread(adcvLoop, adcv, CAMY_ID, &semaphores);
114         pthread_setname_np(adcvThreadY.native_handle(), "adcvThreadY");
115         // adcvThreadX = thread(adcvLoop, adcv, CAMX_ID, &semaphores, &
        logInstance, con);
116         // pthread_setname_np(adcvThreadX.native_handle(), "adcvThreadX");
117         // adcvThreadY = thread(adcvLoop, adcv, CAMY_ID, &semaphores, &
        logInstance, con);
118         // pthread_setname_np(adcvThreadY.native_handle(), "adcvThreadY");
119     }
120
121     // TASKS
122     dataThread = thread(dataLog, adcv, &semaphores, &logInstance, 1); // T6
123     pthread_setname_np(dataThread.native_handle(), "dataThread");
124
125     recThread = thread(recVideo, adcv, &semaphores); // T7
126     pthread_setname_np(recThread.native_handle(), "recThread");
127
128
129     showThread = thread(showImg, adcv, &semaphores); // T9
130     pthread_setname_np(showThread.native_handle(), "showThread");
131
132     if(!args.all) {
133         tcpThread = thread(tcpLoop, adcv, &semaphores, con, 1); // T8
134         pthread_setname_np(tcpThread.native_handle(), "tcpThread");
135     }
136     else {
137         tcpThread = thread(tcpLoop, adcv, &semaphores, con, 3);
138     }
139
140     // adcvLoop(adcv, CAMZ_ID, &semaphores, &logInstance, con);
141     adcvLoop(adcv, CAMZ_ID, &semaphores);
```

```
142
143     if (args.all){
144         adcvThreadX.join();
145         adcvThreadY.join();
146     }
147
148     tcpThread.join();
149     showThread.join();
150     dataThread.join();
151     recThread.join();
152
153     return ADCV_EXIT_SUCCESS;
154 }
155
156 void argumentParser(ARGS *args, int argc, const char** argv){
157     string arg;
158     args->z = true; // DEFAULT
159     for(int counter = 1; counter < argc; counter++){
160         arg = argv[counter];
161         if(arg.compare("-a") == 0 ) args->all = true;
162         else if(arg.compare("-p") == 0 ) args->paralellization = true;
163         else if(arg.compare("-r") == 0 ) args->rec = true;
164         else if(arg.compare("-v") == 0 ) args->video = true;
165         else {
166           if(arg.find("_Z_") != string::npos)
167               args->z_video = arg;
168           if(arg.find("_Y_") != string::npos)
169               args->y_video = arg;
170           if(arg.find("_X_") != string::npos)
171               args->x_video = arg;
172         }
173     }
174     if(argc == 1) args->z = true;
175 }
176
177 void initTCPServerConnection(TCPServer **srv, unsigned short port) {
178     try{
179       (*srv) = new TCPServer(port);
180     }
181     catch(runtime_error *e) {
182         cerr << e->what() << endl;
183         exit(1);
184     }
185 }
186
187 Connection* getConnection(TCPServer **srv){
188     Connection* con = NULL;
189     try {
190         con = (*srv)->accept_wait();
191     }
192     catch(runtime_error *e) {
193         cerr << e->what() << endl;
```

```
194     }
195     if (*srv != NULL) {
196        delete *srv;
197        *srv = NULL;
198     }
199     return con;
200 }
201 
202 /*
203 *    ADCV implementation
204 */
205 
206 void adcvLoop(ADCV *adcv, int cam, Semaphores *semControl) {
207     double frameTimeADCV = 0.0, loopTimeADCV = 0.0;
208     auto start = chrono::high_resolution_clock::now();
209     auto end = chrono::high_resolution_clock::now();
210 
211     double end_frame = 0.0, end_attitude = 0.0, loop_time = 0.0;
212 
213     bool frameOk = true;
214     AdcvTcpData adcvTcpData;
215     JSON toLog;
216 
217     semControl->changeADCVThreadCount(+1);
218     auto start_loop = chrono::high_resolution_clock::now();
219 
220     #if DETERMINE_RTS_TIME
221         ofstream MyFile( std::string(BASE_DIR_RTS)  + "t1_t5_" + std::to_string(
    cam) + ".txt");
222         MyFile << "Data: " << std::string(adcv->dt);
223     #endif
224 
225     while(c != ESC_KEY && adcv->camStatus(cam)){
226         #if DETERMINE_RTS_TIME
227             start_loop = chrono::high_resolution_clock::now();
228         #endif
229         semControl->beginADCVLoop();
230         /// ADCV
231         #if DETERMINE_RTS_TIME
232             start = chrono::high_resolution_clock::now();
233         #endif
234 
235         frameOk = adcv->getFrame(cam);
236         #if DETERMINE_RTS_TIME
237             end_frame = chrono::duration_cast<chrono::nanoseconds>( chrono::
    high_resolution_clock::now() - start).count() * 1e-6;
238         #endif
239 
240         if(frameOk){
241               #if DETERMINE_RTS_TIME
242                   start = chrono::high_resolution_clock::now();
243               #endif
```

```
244
245             adcv->determAttitude(cam);
246
247             #if DETERMINE_RTS_TIME
248                 end_attitude = chrono::duration_cast<chrono::nanoseconds>(
    chrono::high_resolution_clock::now() - start).count() * 1e-6;
249             #endif
250
251             if(cam == CAMZ_ID) {
252                 c = (char)cv::waitKey(1);
253                 semControl->setQuitChar(c);
254             }
255
256             #if DETERMINE_RTS_TIME
257                 end = chrono::high_resolution_clock::now();
258                 loop_time = chrono::duration_cast<chrono::nanoseconds>(end -
    start_loop).count() * 1e-6;
259                 MyFile << end_frame << ";" << end_attitude << ";" << loop_time
    << endl;
260             #endif
261
262             semControl->finishADCVLoop(cam); // Sync all Cams
263             end = chrono::high_resolution_clock::now();
264             loopTimeADCV = chrono::duration_cast<chrono::nanoseconds>(end -
    start_loop).count() * 1e-6; // milliseconds
265             if( loopTimeADCV < CYCLE_TIME) usleep((CYCLE_TIME - loopTimeADCV)
    *1000); // IDLE
266
267         }else{
268             c = ESC_KEY;
269             semControl->setQuitChar(c);
270         }
271     }
272
273     #if DETERMINE_RTS_TIME
274         MyFile.close();
275     #endif
276
277     if(cam == CAMZ_ID) semControl->setQuitChar(c);
278     cout << "Exiting camera = " << cam << endl;
279
280     semControl->changeADCVThreadCount(-1);
281 }
282
283 void dataLog(ADCV *adcv, Semaphores *semControl, JSONLogging *log, int numCam){
    // Task 6
284     JSON toLog;
285     auto start = chrono::high_resolution_clock::now();
286     auto end = chrono::high_resolution_clock::now();
287     double loopTimeData = 0;
288     bool skip = true;
289     semControl->changeADCVDataThreadCount(+1);
```

```
290
291     #if DETERMINE_RTS_TIME
292         ofstream MyFile( std::string(BASE_DIR_RTS) + "t6.txt");
293         MyFile << "Date: " << std::string(adcv->dt);
294     #endif
295
296     while(c != ESC_KEY){
297         // cout << "datalog" << endl;
298         semControl->beginTCPLoop();
299         #if DETERMINE_RTS_TIME
300             start = chrono::high_resolution_clock::now();
301         #endif
302         if(!skip){
303             // cout << "datalog" << endl;
304             toLog = JSON();
305             toLog["cameraId"] = CAMZ_ID;
306             toLog["z"] = 1.1f;
307             toLog["y"] = 2.2f;
308             toLog["x"] = 3.3f;
309             log->adcvLogFrame(toLog);
310         }else skip = false;
311
312         #if DETERMINE_RTS_TIME
313             end = chrono::high_resolution_clock::now();
314             loopTimeData = chrono::duration_cast<chrono::nanoseconds>(end - start
    ).count() * 1e-6; // milliseconds
315             MyFile << loopTimeData << endl;
316         #endif
317
318         semControl->finishTCPLoop();
319     }
320     semControl->changeADCVDataThreadCount(-1);
321     #if DETERMINE_RTS_TIME
322         MyFile.close();
323     #endif
324 }
325
326 void recVideo(ADCV *adcv, Semaphores *semControl){ // Task 7
327     auto start = chrono::high_resolution_clock::now();
328     auto end = chrono::high_resolution_clock::now();
329     double loopTimeRec = 0;
330     bool skip = true;
331     semControl->changeADCVDataThreadCount(+1);
332
333     #if DETERMINE_RTS_TIME
334         ofstream MyFile( std::string(BASE_DIR_RTS) + "t7.txt");
335         MyFile << "Date: " << std::string(adcv->dt);
336     #endif
337
338     while(c != ESC_KEY){
339         semControl->beginTCPLoop();
340         start = chrono::high_resolution_clock::now();
```

```
341         if(!skip){
342             adcv->recVideo();
343         }else skip = false;
344 
345         #if DETERMINE_RTS_TIME
346             end = chrono::high_resolution_clock::now();
347             loopTimeRec = chrono::duration_cast<chrono::nanoseconds>(end - start)
    .count() * 1e-6; // milliseconds
348             MyFile << loopTimeRec << endl;
349         #endif
350 
351         semControl->finishTCPLoop();
352     }
353     semControl->changeADCVDataThreadCount(-1);
354     #if DETERMINE_RTS_TIME
355         MyFile.close();
356     #endif
357 }
358 
359 void showImg(ADCV *adcv, Semaphores *semControl){ // Task9
360     auto end = chrono::high_resolution_clock::now();
361     auto start = chrono::high_resolution_clock::now();
362     double loopTimeShow = 0;
363     bool skip = true;
364     semControl->changeADCVDataThreadCount(+1);
365 
366     #if DETERMINE_RTS_TIME
367         ofstream MyFile( std::string(BASE_DIR_RTS) + "t9.txt");
368         MyFile << "Date: " << std::string(adcv->dt);
369     #endif
370 
371     while(c != ESC_KEY){
372         semControl->beginTCPLoop();
373 
374         #if DETERMINE_RTS_TIME
375             start = chrono::high_resolution_clock::now();
376         #endif
377 
378         if(!skip){
379             adcv->showImage();
380         }else skip = false;
381 
382         #if DETERMINE_RTS_TIME
383             end = chrono::high_resolution_clock::now();
384             loopTimeShow = chrono::duration_cast<chrono::nanoseconds>(end - start
    ).count() * 1e-6; // milliseconds
385             MyFile << loopTimeShow << endl;
386         #endif
387         semControl->finishTCPLoop();
388     }
389     semControl->changeADCVDataThreadCount(-1);
390 
```

```
391     #if DETERMINE_RTS_TIME
392         MyFile.close();
393     #endif
394 }
395 
396 void tcpLoop(ADCV *adcv, Semaphores *semControl, Connection *con, int numCam) {
    // Task 8
397     if (con == NULL) {
398         cerr << "ãConexo ãno instanciada." << endl;
399         exit(TCP_LOOP_NULL_CONNECTION_INSTANCE);
400     }
401     double loopTimeTCP = 0.0, cycleTimeTCP = 0.0;
402     auto start = chrono::high_resolution_clock::now();
403     auto start_t = chrono::high_resolution_clock::now();
404     auto end = chrono::high_resolution_clock::now();
405 
406     AdcvTcpData adcvTcpData;
407     memset(&adcvTcpData, 0, sizeof(AdcvTcpData));
408 
409     semControl->changeADCVDataThreadCount(+1);
410 
411     #if DETERMINE_RTS_TIME
412         ofstream MyFile( std::string(BASE_DIR_RTS) + "t8.txt");
413         MyFile << "Date: " << std::string(adcv->dt);
414     #endif
415 
416     for(adcvTcpData.counter = 0; semControl->changeADCVThreadCount(0) > 0;
    adcvTcpData.counter++) {
417         start = chrono::high_resolution_clock::now();
418 
419         if (c == ESC_KEY) break;
420 
421         semControl->beginTCPLoop();
422 
423         if (numCam == 3) {
424             adcvTcpData.x = adcv->attitude[8][2];
425             adcvTcpData.y = adcv->attitude[7][2];
426             adcvTcpData.z = adcv->attitude[6][2];
427         }
428         else {
429             adcvTcpData.x = adcv->attitude[6][0];
430             adcvTcpData.y = adcv->attitude[6][1];
431             adcvTcpData.z = adcv->attitude[6][2];
432         }
433         adcvTcpData.shouldExit = 0;
434         con->sendAdcvTcpData(&adcvTcpData);
435 
436         end = chrono::high_resolution_clock::now();
437         loopTimeTCP = chrono::duration_cast<chrono::nanoseconds>(end - start).
    count() * 1e-6; // milliseconds
438         MyFile << adcv->attitude[8][0] << ";" << adcv->attitude[8][1] << ";" <<
    adcv->attitude[8][2] << ";" << adcv->attitude[7][0] << ";" << adcv->attitude
```

```
    [7][1] << ";" << adcv->attitude[7][2] << ";" << adcv->attitude[6][0] << ";" <<
     adcv->attitude[6][1] << ";" << adcv->attitude[6][2] << ";" << endl;
439         cout << adcvTcpData.counter << ";" << loopTimeTCP << endl;
440         semControl->finishTCPLoop();
441
442     }
443     end = chrono::high_resolution_clock::now();
444     loopTimeTCP = chrono::duration_cast<chrono::nanoseconds>(end - start_t).count
    () * 1e-6; // milliseconds
445     cout << "Tempo émdio: " << (loopTimeTCP / adcvTcpData.counter)  << endl;
446     //memset(&adcvTcpData, 0, sizeof(AdcvTcpData));
447     adcvTcpData.shouldExit = 1;
448     con->sendAdcvTcpData(&adcvTcpData);
449     cout << "exiting" << endl;
450
451     semControl->setQuitChar(c);
452     semControl->changeADCVDataThreadCount(-1);
453
454     #if DETERMINE_RTS_TIME
455         MyFile.close();
456     #endif
457
458     if (con != NULL) {
459         delete con;
460         con = NULL;
461     }
462 }
463
464 void removeSemaphores(int signum) {
465     if ((globalSemId != 0) && (semctl(globalSemId, 0, IPC_RMID) < 0)) {
466         cerr << "handleSignals: " << strerror(errno) << endl;
467         exit(ADCV_SEMCTL_FAIL);
468     }
469     cout << "áSemforos removidos." << endl;
470 }
```

II.2 Exemplo das funções dos semáforos retiradas do arquivo *semaphores.cpp*

```
1
2 void Semaphores::beginADCVLoop(void) {
3     // Incrementa ásemforo
4     struct sembuf threadLocalSemOp;
5     threadLocalSemOp.sem_num = ADCV_LOOP_SEM;
6     threadLocalSemOp.sem_op  = 1;
7     threadLocalSemOp.sem_flg = SEM_UNDO;
8     if (semop(ADCVSemId, &threadLocalSemOp, 1) < 0) {
9         cerr << "Semaphores::beginADCVLoop: " << strerror(errno) << endl;
10        exit(ADCV_SEMOP_FAIL);
```

```
    }
}

void Semaphores::finishADCVLoop(int camId) {
    struct sembuf threadLocalSemOp;
    // Decrementa ásemforo
    threadLocalSemOp.sem_num = ADCV_LOOP_SEM;
    threadLocalSemOp.sem_op  = -1;
    threadLocalSemOp.sem_flg = SEM_UNDO;
    if (semop(ADCVSemId, &threadLocalSemOp, 1) < 0) {
        cerr << "Semaphores::finishADCVLoop: " << strerror(errno) << endl;
        exit(ADCV_SEMOP_FAIL);
    }
    // Espera por zero, bloqueando as threads
    threadLocalSemOp.sem_num = ADCV_LOOP_SEM;
    threadLocalSemOp.sem_op  = 0;
    threadLocalSemOp.sem_flg = SEM_UNDO;
    if (semop(ADCVSemId, &threadLocalSemOp, 1) < 0) {
        cerr << "Semaphores::finishADCVLoop: " << strerror(errno) << endl;
        exit(ADCV_SEMOP_FAIL);
    }
    // Libera thread TCP
    if (camId == CAMZ_ID && ADCVDataCount != 0) {
        threadLocalSemOp.sem_num = ADCV_TCP_LOOP_SEM;
        threadLocalSemOp.sem_op  = -1;
        threadLocalSemOp.sem_flg = SEM_UNDO;
        for(int i = 0; i < changeADCVDataThreadCount(0); i++)
        {
            if (semop(ADCVSemId, &threadLocalSemOp, 1) < 0) {
                cerr << "Semaphores::finishADCVLoop: " << strerror(errno) << endl
    ;
                exit(ADCV_SEMOP_FAIL);
            }
        }
    }
}


void Semaphores::beginDataLoop(void) {
    // Espera por zero
    struct sembuf threadLocalSemOp;
    threadLocalSemOp.sem_num = ADCV_TCP_LOOP_SEM;
    threadLocalSemOp.sem_op  = 0;
    threadLocalSemOp.sem_flg = SEM_UNDO;
    if (semop(ADCVSemId, &threadLocalSemOp, 1) < 0) {
        cerr << "Semaphores::beginTCPLoop: " << strerror(errno) << endl;
        exit(ADCV_SEMOP_FAIL);
    }
}

void Semaphores::finishDataLoop(void) {
    // Bloqueia as threads de dados
```

```
    struct sembuf threadLocalSemOp;
    threadLocalSemOp.sem_num = ADCV_TCP_LOOP_SEM;
    threadLocalSemOp.sem_op  = 1;
    threadLocalSemOp.sem_flg = SEM_UNDO;
    if (semop(ADCVSemId, &threadLocalSemOp, 1) < 0) {
        cerr << "Semaphores::beginTCPLoop: " << strerror(errno) << endl;
        exit(ADCV_SEMOP_FAIL);
    }
}
```

II.3 Código para análise e plot da seção 4.2.2.

```
import argparse
import numpy as np
import matplotlib.pyplot as plt

TIME_TH_MIN = 20
TIME_TH_MAX = 80

PERCENTAGE_TH = 30
ANGLE_TH = 360

computed_data = {
    "Num_samples": 0,
    "Time_adcv": {"Mean": 0, "Std": 0, "Median": 0, "HWM": 0, "pWCRT": 0},
    "Time_rpi": {"Mean": 0, "Std": 0, "Median": 0, "HWM": 0, "pWCRT": 0},
    "Cam_z": {"Mean": 0, "Std": 0},
    "Cam_y": {"Mean": 0, "Std": 0},
    "Cam_x": {"Mean": 0, "Std": 0},
}

computed_data_lb = {
    "Num_samples": 0,
    "Time_adcv": {"Mean": 0, "Std": 0, "Median": 0, "HWM": 0, "pWCRT": 0},
    "Time_rpi": {"Mean": 0, "Std": 0, "Median": 0, "HWM": 0, "pWCRT": 0},
    "Cam_z": {"Mean": 0, "Std": 0},
    "Cam_y": {"Mean": 0, "Std": 0},
    "Cam_x": {"Mean": 0, "Std": 0},
}


def load_data(file=None):
    f = open(file, "r")
    data = []
    for line in f.readlines():
        elements = line.split("; ")
        if len(elements) == 1:
            elements = line.split(";")
        if (
            len(elements) > 1
```

```
        ):
            data.append(np.fromstring(line, dtype=float, sep="; "))
    f.close()
    return np.array(data)


def correct_data(time_adcv, time_rpi, adcv, rpi):
    temp_adcv = []
    temp_rpi = []
    temp_time_rpi = []
    temp_time_adcv = []

    for i in range(0, len(rpi[:, 0])):
        if (
            (rpi[i, 0] < 360 and rpi[i, 0] > 0)
            and (rpi[i, 1] < 360 and rpi[i, 1] > -360)
            and (rpi[i, 2] < 360 and rpi[i, 2] > -360)
            and (float(time_adcv[i]) > TIME_TH_MIN)
            and (float(time_adcv[i]) < TIME_TH_MAX)
            and (float(time_rpi[i]) > TIME_TH_MIN)
            and (float(time_rpi[i]) < TIME_TH_MAX)
        ):
            temp_adcv.append(adcv[i, :])
            temp_rpi.append(rpi[i, :])
            temp_time_rpi.append(time_rpi[i])
            temp_time_adcv.append(time_adcv[i])

    temp_adcv = np.array(temp_adcv)
    temp_rpi = np.array(temp_rpi)
    temp_time_rpi = np.array(temp_time_rpi)
    temp_time_adcv = np.array(temp_time_adcv)

    return temp_time_adcv, temp_time_rpi, temp_adcv, temp_rpi

def GP(data=[], task_index=0, epsilon=1, th=1, scale=1, plot_flag=False):
    pdf = np.zeros((len(data), 1))
    cdf = np.zeros((len(data), 1))
    cdf_1 = np.zeros((len(data), 1))
    # if(epsilon > 0):

    for i in range(0, len(data)):
        # CDF DA GP
        cdf[i] = 1 - (1 + epsilon * ((data[i] - th) / scale)) ** (-1 / epsilon)
        # 1 - CDF da GP
        cdf_1[i] = 1 - cdf[i]
        # PDF DA GP
        pdf[i] = (1 / scale) * (1 + epsilon * ((data[i] - th) / scale)) ** (
            -1 * (1 + epsilon) / epsilon
        )

    print("Data = {}, pdf_gp = {}".format(data[-1], pdf[-1]))
    if plot_flag:
```

```
91          plt.plot(data, pdf, color="blue", marker="*")
92          plt.title("PDF DA GP")
93          plt.xlabel("Tempo de resposta [ms]")
94          plt.ylabel("H(X)")
95          plt.savefig(args["output"] + "t" + str(task_index) + "_pdf_gp" + ".png")
96          plt.show()
97
98      print("Data = {}, cdf_gp = {}".format(data[-1], cdf[-1]))
99      if plot_flag:
100         plt.plot(data, cdf, color="red", marker="*")
101         plt.title("CDF DA GP")
102         plt.xlabel("Tempo de resposta [ms]")
103         plt.ylabel("H(X)")
104         plt.savefig(args["output"] + "t" + str(task_index) + "_cdf_gp" + ".png")
105         plt.show()
106
107     print("Data = {}, 1-cdf_gp = {}".format(data[-1], cdf_1[-1]))
108     if plot_flag:
109         plt.plot(data, cdf_1, color="black", marker="*")
110         plt.title("1 - CDF DA GP")
111         plt.xlabel("Tempo de resposta [ms]")
112         plt.ylabel("H(X)")
113         plt.savefig(args["output"] + "t" + str(task_index) + "_1-cdf_gp" + ".png"
    )
114         plt.show()
115
116     return pdf[-1]
117
118 def POT(data, th):
119     pot = [i for i in data if i >= th]
120     time_list  = np.zeros((len(pot),1))
121     for i in range(0, len(pot)):
122         time_list[i] = i
123
124     plt.scatter(time_list, pot, color='blue', marker='*')
125     plt.title('Tempo de resposta do POT')
126     plt.xlabel('Amostra')
127     plt.ylabel('Tempo de resposta [ms]')
128     # plt.xlim(0.5, 4.5)
129     plt.show()
130     return pot
131
132 def pWCRT(time=[], task=0):
133     th = np.mean(time) + 2*np.std(time)
134     a = np.sort(time)
135     pot = POT(a, th)
136     scale = 1
137     # epsilon = int(input("Type task epsilon = "))
138     epsilon = 1000
139     pot = np.sort(pot)
140     print(
```

```
141         "Compute GP with parameters:\n\tData_th={}\n\tScale={}\n\tEpsilon={}".
    format(
142             th, scale, epsilon
143         )
144     )
145     pwcrt = GP(pot, task, epsilon, th, scale, False)
146     return pwcrt
147 
148 def compute_stats(time_adcv=[], time_rpi=[], data={}):
149     data["Num_samples"] = len(time_adcv)
150     data["Time_adcv"]["Mean"] = np.mean(time_adcv)
151     data["Time_adcv"]["Std"] = np.std(time_adcv)
152     data["Time_adcv"]["Median"] = np.median(time_adcv)
153     data["Time_adcv"]["HWM"] = np.amax(time_adcv)
154     data["Time_adcv"]["pWCRT"] = 1.2*np.amax(time_adcv)
155 
156     data["Time_rpi"]["Mean"] = np.mean(time_rpi)
157     data["Time_rpi"]["Std"] = np.std(time_rpi)
158     data["Time_rpi"]["Median"] = np.median(time_rpi)
159     data["Time_rpi"]["HWM"] = np.amax(time_rpi)
160     data["Time_rpi"]["pWCRT"] = 1.2*np.amax(time_rpi)
161 
162     print("Min_adcv = {:.3f}, Max_adcv = {:.3f}, Min_rpi = {:.3f}, Max_rpi = {:.3
    f}".format(np.amin(time_adcv), np.amax(time_adcv), np.amin(time_rpi), np.amax(
    time_rpi)))
163     print("Mean_adcv = {:.3}, Mean_rpi = {:.3}".format(np.mean(time_adcv), np.
    mean(time_rpi)))
164     print("ADCV - Percentage greather than {}ms: {:.3f}%".format(PERCENTAGE_TH,
    np.count_nonzero((time_adcv > PERCENTAGE_TH))/time_adcv.size*100))
165     print("RPI - Percentage greather than {}ms: {:.3f}%".format(PERCENTAGE_TH, np
    .count_nonzero((time_rpi > PERCENTAGE_TH))/time_rpi.size*100))
166 
167     return data
168 
169 def plot(data = {}, data_lb = {}):
170     # data["Num_samples"]
171     # data_lb["Num_samples"]
172     time_stats = [
173             [data_lb["Time_adcv"]["Median"],data_lb["Time_adcv"]["Mean"], data_lb
    ["Time_adcv"]["HWM"], data_lb["Time_adcv"]["pWCRT"]],
174             [data_lb["Time_rpi"]["Median"], data_lb["Time_rpi"]["Mean"], data_lb[
    "Time_rpi"]["HWM"], data_lb["Time_rpi"]["pWCRT"]],
175             [data["Time_adcv"]["Median"],data["Time_adcv"]["Mean"], data["
    Time_adcv"]["HWM"], data["Time_adcv"]["pWCRT"]],
176             [data["Time_rpi"]["Median"], data["Time_rpi"]["Mean"], data["Time_rpi
    "]["HWM"], data["Time_rpi"]["pWCRT"]],
177         ]
178     #y_err = [data["Time_adcv"]["Std"], data["Time_rpi"]["Std"], data_lb["
    Time_adcv"]["Std"],data_lb["Time_rpi"]["Std"]]
179     index = np.arange(len(time_stats))
180     data_name = ["Mediana", "éMdia", "HWM", "pWCRT"]
181     exp_name = ["Adcv_lb", "Rpi_lb", "Adcv", "Rpi"]
```

```
182     x = np.arange(len(data_name))*4+1.5
183     width = 1
184     fig, ax = plt.subplots()
185     for i in range(0, len(data_name)):
186         rect = ax.bar(4*i, time_stats[i][0], width, color = "c", label=data_name[
    i])
187         ax.annotate('{:.2f}'.format(time_stats[i][0]),
188                     xy=(4*i, time_stats[i][0]),
189                     xytext=(0, 5),
190                     textcoords="offset points",
191                     ha='center', va='bottom', fontsize=7)
192 
193         rect = ax.bar(1+4*i, time_stats[i][1], width, color = "r", label=
    data_name[i])
194         ax.annotate('{:.2f}'.format(time_stats[i][1]),
195                     xy=(1+4*i, time_stats[i][1]),
196                     xytext=(0, 10),
197                     textcoords="offset points",
198                     ha='center', va='bottom', fontsize=7)
199 
200         rects = ax.bar(2+4*i, time_stats[i][2], width, color = "g", label=
    data_name[i])
201         ax.annotate('{:.2f}'.format(time_stats[i][2]),
202                     xy=(2+4*i, time_stats[i][2]),
203                     xytext=(0, 5),
204                     textcoords="offset points",
205                     ha='center', va='bottom', fontsize=7)
206 
207         rects = ax.bar(3+4*i, time_stats[i][3], width, color = "b", label=
    data_name[i])
208         ax.annotate('{:.2f}'.format(time_stats[i][3]),
209                     xy=(3+4*i, time_stats[i][3]),
210                     xytext=(0, 5),
211                     textcoords="offset points",
212                     ha='center', va='bottom', fontsize=7)
213 
214     plt.legend(labels = data_name)
215     plt.xticks(index, data_name)
216     ax.set_title('Medidas íestatsticas da çãcomunicao ADCV-Rpi e ADCV-Loopback')
217     ax.set_ylabel('Tempo de resposta [ms]')
218     ax.set_xticks(x)
219     ax.set_xticklabels(exp_name)
220     plt.show()
221 
222 def main():
223     global computed_data, computed_data_lb
224     data_adcv = load_data(args["adcv"])
225     data_rpi = load_data(args["rpi"])
226     data_adcv_lb = load_data(args["adcv_loopback"])
227     data_rpi_lb = load_data(args["rpi_loopback"])
228 
229     time_adcv, time_rpi, adcv, rpi = correct_data(
```

```
        time_adcv=data_adcv[:, 1],
        time_rpi=data_rpi[:, 1],
        adcv=data_rpi[:, 2:5],
        rpi=data_rpi[:, 5:8],
    )
    time_adcv_lb, time_rpi_lb, adcv_lb, rpi_lb = correct_data(
        time_adcv=data_adcv_lb[:, 1],
        time_rpi=data_rpi_lb[:, 1],
        adcv=data_rpi_lb[:, 2:5],
        rpi=data_rpi_lb[:, 2:5],
    )
    print("Experiment ADCV-RPI")
    computed_data = compute_stats(time_adcv=time_adcv, time_rpi=time_rpi, data=
    computed_data)
    print("\n\nExperiment in LoopBack")
    computed_data_lb = compute_stats(time_adcv=time_adcv_lb, time_rpi=time_rpi_lb
    , data=computed_data_lb)
    plot(data = computed_data, data_lb = computed_data_lb)

if __name__ == "__main__":
    ap = argparse.ArgumentParser()
    ap.add_argument(
        "-al",
        "--adcv_loopback",
        type=str,
        default="../../time_rts/loopback/adcv_loop_time.txt",
        help="Data file",
    )
    ap.add_argument(
        "-rl",
        "--rpi_loopback",
        type=str,
        default="../../time_rts/loopback/rpi_loop_time.txt",
        help="Data file",
    )
    ap.add_argument(
        "-a",
        "--adcv",
        type=str,
        default="../../time_rts/rpi/adcv_loop.txt",
        help="Data file",
    )

    ap.add_argument(
        "-r",
        "--rpi",
        type=str,
        default="../../time_rts/rpi/rpi_loop.txt",
        help="Data file",
    )
    ap.add_argument(
        "-o",
```

```
        "--output",
        type=str,
        default="../../time_rts/output_data/",
        help="Data file",
    )
    args = vars(ap.parse_args())
    main()
```

II.4 Código de análise e plot da sessão 5.2 *computeAdcvAccuracy.py*

```
import argparse
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
from scipy.stats import norm
from math import ceil
import mpl_toolkits.mplot3d.axes3d as p3
import matplotlib.animation as animation

from sklearn.metrics import mean_squared_error as mse

TIME_TH_MIN = 20
TIME_TH_MAX = 100

PERCENTAGE_TH = 66
ANGLE_TH = 360


def load_data(file=None):
    f = open(file, "r")
    data = []
    for line in f.readlines():
        elements = line.split("; ")
        if len(elements) == 1:
            elements = line.split(";")
        if (
            len(elements) > 1
            and float(elements[1]) > TIME_TH_MIN
            and float(elements[1]) < TIME_TH_MAX
        ):
            data.append(np.fromstring(line, dtype=float, sep="; "))
    f.close()
    return np.array(data)


def POT(data, th):
    pot = [i for i in data if i >= th]
    time_list = np.zeros((len(pot), 1))
```

```
    for i in range(0, len(pot)):
        time_list[i] = i

    # plt.scatter(time_list, pot, color='blue', marker='*')
    # plt.title('Tempo de resposta do POT')
    # plt.xlabel('Amostra')
    # plt.ylabel('Tempo de resposta [ms]')
    # # plt.xlim(0.5, 4.5)
    # plt.show()
    return pot, time_list


def GP(data=[], task_index=0, epsilon=1, th=1, scale=1, plot_flag=False):
    pdf = np.zeros((len(data), 1))
    cdf = np.zeros((len(data), 1))
    cdf_1 = np.zeros((len(data), 1))
    # if(epsilon > 0):

    for i in range(0, len(data)):
        # CDF DA GP
        cdf[i] = 1 - (1 + epsilon * ((data[i] - th) / scale)) ** (-1 / epsilon)
        # 1 - CDF da GP
        cdf_1[i] = 1 - cdf[i]
        # PDF DA GP
        pdf[i] = (1 / scale) * (1 + epsilon * ((data[i] - th) / scale)) ** (
            -1 * (1 + epsilon) / epsilon
        )

    print("Data = {}, pdf_gp = {}".format(data[-1], pdf[-1]))
    if plot_flag:
        plt.plot(data, pdf, color="blue", marker="*")
        plt.title("PDF DA GP")
        plt.xlabel("Tempo de resposta [ms]")
        plt.ylabel("H(X)")
        plt.savefig(args["output"] + "t" + str(task_index) + "_pdf_gp" + ".png")
        plt.show()

    print("Data = {}, cdf_gp = {}".format(data[-1], cdf[-1]))
    if plot_flag:
        plt.plot(data, cdf, color="red", marker="*")
        plt.title("CDF DA GP")
        plt.xlabel("Tempo de resposta [ms]")
        plt.ylabel("H(X)")
        plt.savefig(args["output"] + "t" + str(task_index) + "_cdf_gp" + ".png")
        plt.show()

    print("Data = {}, 1-cdf_gp = {}".format(data[-1], cdf_1[-1]))
    if plot_flag:
        plt.plot(data, cdf_1, color="black", marker="*")
        plt.title("1 - CDF DA GP")
        plt.xlabel("Tempo de resposta [ms]")
        plt.ylabel("H(X)")
```

```
91          plt.savefig(args["output"] + "t" + str(task_index) + "_1-cdf_gp" + ".png"
    )
92          plt.show()
93
94
95  def get_statistics(data=[], plot=False):
96      print("Num : ", len(data))
97      print("Mean: ", np.mean(data))
98      print("Median: ", np.median(data))
99      print("STD: ", np.std(data))
100     print("Var: ", np.var(data))
101     print("Min: ", np.amin(data))
102     print("Max: ", np.amax(data))
103     if plot:
104         num_samples = [x for x in range(0, len(data))]
105         plt.scatter(num_samples, data, color="blue", marker="*")
106         plt.title("Tempo de resposta")
107         plt.xlabel("Amostra")
108         plt.ylabel("Tempo de resposta [ms]")
109         plt.show()
110         # plt.savefig(args["file"] + ".png")
111         plt.close()
112     return np.mean(data), np.std(data)
113
114
115 def analyse_rt(time=[], task=0):
116
117     mean_t, std_t = get_statistics(data=time, plot=False)
118     print(
119         "Percentage greather than {}ms: {:.3f}%".format(
120             PERCENTAGE_TH, np.count_nonzero((time > PERCENTAGE_TH)) / time.size *
     100
121         )
122     )
123     print(
124         "Number of samples greather than {}ms: {}".format(
125             PERCENTAGE_TH, np.count_nonzero((time > PERCENTAGE_TH))
126         )
127     )
128
129     th = np.mean(time) + np.std(time)
130     a = np.sort(time)
131     pot, time_list = POT(a, th)
132     scale = 1
133     # epsilon = int(input("Type task epsilon = "))
134     epsilon = 1000
135     pot = np.sort(pot)
136     print(
137         "Compute GP with parameters:\n\tData_th={}\n\tScale={}\n\tEpsilon={}".
    format(
138             th, scale, epsilon
139         )
```

```
    )
    GP(pot, task, epsilon, th, scale, False)


def correct_data_rpi(adcv, rpi):
    temp_adcv = []
    temp_rpi = []

    for i in range(0, len(rpi[:, 0])):
        if (
            (rpi[i, 0] < 360 and rpi[i, 0] > 0)
            and (rpi[i, 1] < 360 and rpi[i, 1] > 0)
            and (rpi[i, 2] < 360 and rpi[i, 2] > 0)
        ):
            temp_adcv.append(adcv[i, :])
            temp_rpi.append(rpi[i, :])

    temp_adcv = np.array(temp_adcv)
    temp_rpi = np.array(temp_rpi)
    return temp_adcv, temp_rpi


def analyse_static_accuracy(adcv=[], rpi=[]):

    adcv, rpi = correct_data_rpi(adcv, rpi)
    print("\n\n###### ADCV_Z")
    mean_a_z, std_a_z = get_statistics(adcv[:, 0])
    print("\n### ADCV_Y")
    mean_a_y, std_a_y = get_statistics(adcv[:, 1])
    print("\n### ADCV_X")
    mean_a_x, std_a_x = get_statistics(adcv[:, 2])

    print("\n\n###### RPI_Z")
    mean_r_z, _ = get_statistics(rpi[:, 0])
    print("\n### RPI_Y")
    mean_r_y, _ = get_statistics(rpi[:, 1])
    print("\n### RPI_X")
    mean_r_x, _ = get_statistics(rpi[:, 2])

    # t2, p2 = stats.ttest_ind(a,b)
    offset_z = abs((mean_a_z - mean_r_z + 180) % 360 - 180)
    offset_y = abs((mean_a_y - mean_r_y + 180) % 360 - 180)
    offset_x = abs((mean_a_x - mean_r_x + 180) % 360 - 180)
    print(360 - rpi[:, 0])
    print(adcv[:, 0])
    print(max(adcv[:, 0]) - min(adcv[:, 0]))
    print(
        "\nComputing Offsets:\n\tZ = {}, ERR = {}\n\tY = {}, ERR = {}\n\tX = {},
    ERR = {}".format(
            offset_z, std_a_z, offset_y, std_a_y, offset_x, std_a_x
        )
    )
```

```
    mse_z = mse(360 - rpi[:, 0], adcv[:, 0], squared=False)
    rmse_z = mse(360 - rpi[:, 0], adcv[:, 0], squared=True)
    nrmse_z = mse(360 - rpi[:, 0], adcv[:, 0], squared=True) / (
        max(adcv[:, 0]) - min(adcv[:, 0])
    )
    print(
        "Errors in Z:\n\tMSE = {},\n\tRMSE = {},\n\tNRMSE = {}\n\n".format(
            mse_z, rmse_z, nrmse_z
        )
    )
    mse_y = mse(rpi[:, 1], offset_y + adcv[:, 1], squared=False)
    rmse_y = mse(rpi[:, 1], offset_y + adcv[:, 1], squared=True)
    nrmse_y = mse(rpi[:, 1], offset_y + adcv[:, 1], squared=True) / (
        max(adcv[:, 1]) - min(adcv[:, 1])
    )
    print(
        "Errors in Y:\n\tMSE = {},\n\tRMSE = {},\n\tNRMSE = {}\n\n".format(
            mse_y, rmse_y, nrmse_y
        )
    )
    mse_x = mse(rpi[:, 2], -1 * adcv[:, 2], squared=False)
    rmse_x = mse(rpi[:, 2], -1 * adcv[:, 2], squared=True)
    nrmse_x = mse(rpi[:, 2], -1 * adcv[:, 2], squared=True) / (
        max(adcv[:, 2]) - min(adcv[:, 2])
    )
    print(
        "Errors in X:\n\tMSE = {},\n\tRMSE = {},\n\tNRMSE = {}\n\n".format(
            mse_x, rmse_x, nrmse_x
        )
    )


def plot_3D(adcv=[], rpi=[]):
    print("PLOTTING 3D!")
    adcv, rpi = correct_data_rpi(adcv, rpi)

    fig = plt.figure()
    ax = plt.axes(projection="3d")
    # print(15.5 + adcv[0,2], adcv[0,1], 360 - adcv[0,0])
    # print(rpi[0,2], rpi[0,1], rpi[0,0])

    for i in range(0, len(adcv[:, 2])):
        ax.plot3D(
            [0, 15.5 + adcv[i, 2]], [0, 3.5 + adcv[i, 1]], [0, 360 - adcv[i, 0]],
     "blue"
        )
    ax.plot3D([0, rpi[0, 2]], [0, rpi[0, 1]], [0, rpi[0, 0]], "red", label="IMU")
    # ax.plot3D(rpi[0,2], rpi[:,1], rpi[:,0], 'blue')
    # ax.scatter3D(adcv[0,2], adcv[0,1], adcv[0,0], c=adcv[0,0], cmap='Greens',
    label='ADCV')
    # ax.scatter3D(rpi[:,2], rpi[:,1], rpi[:,0], c=rpi[:,0], cmap='Blues',label='
    IMU')
```

```
240
241     # ax.set_xlim3d([0.0, 1.0])
242     ax.set_xlabel("X")
243
244     # ax.set_ylim3d([0.0, 1.0])
245     ax.set_ylabel("Y")
246
247     # ax.set_zlim3d([0.0, 1.0])
248     ax.set_zlabel("Z")
249
250     ax.legend(loc="upper left")
251     plt.show()
252
253
254 def plot_2d(adcv=[], rpi=[]):
255     print("PLOTTING 2D: XY, XZ, YZ")
256     adcv, rpi = correct_data_rpi(adcv, rpi)
257     for i in range(0, len(adcv[:, 2])):
258         plt.plot([0, 15.5 + adcv[i, 2]], [0, 3.5 + adcv[i, 1]], "b", label="ADCV"
    )
259         plt.plot([0, rpi[i, 2]], [0, rpi[i, 1]], "r", label="ADCV")
260         if i == 0:
261             plt.legend(loc="upper left")
262     plt.xlabel("X")
263     plt.ylabel("Y")
264     plt.show()
265
266     for i in range(0, len(adcv[:, 2])):
267         plt.plot([0, 15.5 + adcv[i, 2]], [0, 360 - adcv[i, 0]], "b", label="ADCV"
    )
268         plt.plot([0, rpi[i, 2]], [0, rpi[i, 0]], "r", label="ADCV")
269         if i == 0:
270             plt.legend(loc="upper left")
271     plt.xlabel("X")
272     plt.ylabel("Z")
273     plt.show()
274
275     for i in range(0, len(adcv[:, 2])):
276         plt.plot([0, 3.5 + adcv[i, 1]], [0, 360 - adcv[i, 0]], "b", label="ADCV")
277         plt.plot([0, rpi[i, 1]], [0, rpi[i, 0]], "r", label="ADCV")
278         if i == 0:
279             plt.legend(loc="upper left")
280     plt.xlabel("Y")
281     plt.ylabel("Z")
282     plt.show()
283
284
285 def plot_3d_animation(adcv=[], rpi=[]):
286     print("PLOTTING 3D ANIMATION!")
287     adcv, rpi = correct_data_rpi(adcv, rpi)
288     fig = plt.figure()
289     ax = plt.axes(projection="3d")
```

```
290     ax.set_xlabel("X")
291     ax.set_ylabel("Y")
292     ax.set_zlabel("Z")
293 
294     plt.ion()
295     plt.show()
296     for i in range(0, len(adcv[:, 2])):
297         print(rpi[i, 2], rpi[i, 1], rpi[i, 0])
298         print(15.5 + adcv[i, 2], 3.5 + adcv[i, 1], 360 - adcv[i, 0])
299         ax.plot3D(
300             [0, 15.5 + adcv[i, 2]],
301             [0, 3.5 + adcv[i, 1]],
302             [0, 360 - adcv[i, 0]],
303             "blue",
304             label="ADCV",
305         )
306         ax.plot3D([0, rpi[i, 2]], [0, rpi[i, 1]], [0, rpi[i, 0]], "red", label="
    IMU")
307         # ax.plot3D([0, 8.5 + adcv[i,2]], [0, 3.5 + adcv[i,1]], [0, 360 - adcv[i
    ,0] ], 'blue', label='ADCV')
308         plt.pause(0.1)
309         if i == 0:
310             ax.legend(loc="upper left")
311 
312 
313 def plot_angle_by_time(adcv=[], rpi=[]):
314     print("PLOTTING 2D: XY, XZ, YZ")
315     adcv, rpi = correct_data_rpi(adcv, rpi)
316     time = np.arange(0.0, len(adcv[:, 0]), 1) * 33
317 
318     plt.plot(time, 360 - adcv[:, 0], "o", color="b", label="ADCV")
319     plt.plot(time, rpi[:, 0], color="r", label="IMU")
320     plt.xlabel("Tempo [ms]")
321     plt.ylabel("Ângulo °()")
322     plt.show()
323 
324     plt.plot(time, 3.5 + adcv[:, 1], "o", color="b", label="ADCV")
325     plt.plot(time, rpi[:, 1], color="r", label="IMU")
326     plt.xlabel("Tempo [ms]")
327     plt.ylabel("Ângulo °()")
328     plt.show()
329 
330     plt.plot(time, 15.5 + adcv[:, 2], "o", color="b", label="ADCV")
331     plt.plot(time, rpi[:, 2], color="r", label="IMU")
332     plt.xlabel("Tempo [ms]")
333     plt.ylabel("Ângulo °()")
334     plt.show()
335 
336 
337 def plot_accuracy(adcv=[], rpi=[]):
338     print("PLOTTING 2D: XY, XZ, YZ")
339     adcv, rpi = correct_data_rpi(adcv, rpi)
```

```
340     time = np.arange(0.0, len(adcv[:, 0]), 1) * 33
341 
342     ######################### YZ
343 
344     theta = np.linspace(0, 2 * np.pi, 150)
345     radius = mse(rpi[:, 1], 3.5 + adcv[:, 1], squared=False)
346     print("Raio YZ = {}".format(radius))
347     a = radius * np.cos(theta)
348     b = radius * np.sin(theta)
349     figure, axes = plt.subplots(1)
350     axes.plot(rpi[0, 1] + a, rpi[0, 0] + b)
351 
352     plt.plot(3.5 + adcv[:, 1], 360 - adcv[:, 0], "o", color="b", label="ADCV")
353     plt.plot(rpi[:, 1], rpi[:, 0], "*", color="r", label="IMU")
354     plt.title("YZ")
355     plt.xlabel("Tempo [ms]")
356     plt.ylabel("Ângulo °()")
357     plt.grid()
358     plt.show()
359 
360     ######################### XZ
361 
362     theta = np.linspace(0, 2 * np.pi, 150)
363     radius = mse(rpi[:, 2], -1 * adcv[:, 2], squared=False)
364     print("Raio XZ = {}".format(radius))
365     a = radius * np.cos(theta)
366     b = radius * np.sin(theta)
367     figure, axes = plt.subplots(1)
368     axes.plot(rpi[0, 2] + a, rpi[0, 0] + b)
369 
370     plt.plot(15.5 + adcv[:, 2], 360 - adcv[:, 0], "o", color="b", label="ADCV")
371     plt.plot(rpi[:, 2], rpi[:, 0], "*", color="r", label="IMU")
372     plt.title("XZ")
373     plt.xlabel("Tempo [ms]")
374     plt.ylabel("Ângulo °()")
375     plt.grid()
376     plt.show()
377 
378     ######################### XY
379 
380     theta = np.linspace(0, 2 * np.pi, 150)
381     radius = mse(rpi[:, 1], 3.5 + adcv[:, 1], squared=False)
382     print("Raio XY = {}".format(radius))
383     a = radius * np.cos(theta)
384     b = radius * np.sin(theta)
385     figure, axes = plt.subplots(1)
386     axes.plot(rpi[0, 2] + a, rpi[0, 1] + b)
387 
388     plt.plot(15.5 + adcv[:, 2], 3.5 + adcv[:, 1], "o", color="b", label="ADCV")
389     plt.plot(rpi[:, 2], rpi[:, 1], "*", color="r", label="IMU")
390     plt.title("XY")
391     plt.xlabel("Tempo [ms]")
```

```
392     plt.ylabel("Ângulo °()")
393     plt.grid()
394     plt.show()
395 
396 
397 def main():
398     data = load_data(args["dataset"])
399     analyse_rt(time=data[:,1], task=0) # task0 is rpi
400     analyse_static_accuracy(adcv=data[:, 2:5], rpi=data[:, 5:8])
401     # plot_3D(adcv = data[:,2:5], rpi = data[:,5:8])
402     # plot_2d(adcv = data[:,2:5], rpi = data[:,5:8])
403     # plot_3d_animation(adcv = data[:,2:5], rpi = data[:,5:8])
404     plot_angle_by_time(adcv = data[:,2:5], rpi = data[:,5:8])
405     plot_accuracy(adcv=data[:, 2:5], rpi=data[:, 5:8])
406 
407 
408 if __name__ == "__main__":
409     ap = argparse.ArgumentParser()
410     ap.add_argument(
411         "-d",
412         "--dataset",
413         type=str,
414         default="../../time_rts/rpi/rpi_loop.txt",
415         help="Data file",
416     )
417     ap.add_argument(
418         "-o",
419         "--output",
420         type=str,
421         default="../../time_rts/output_data/",
422         help="Data file",
423     )
424     args = vars(ap.parse_args())
425     main()
```